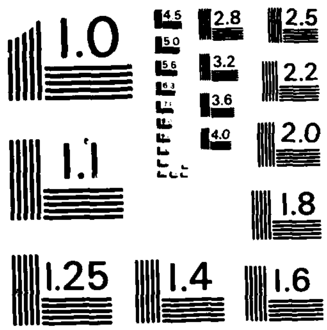
AD-A160 973 C3 EVAL MODEL DEVELOPMENT AND TEST VOLUME 2 PROGRAMMERS 1/3
MANUAL..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA
VA R F ROBINSON ET AL. OCT 85 IDA-P-1882-VOL-2
UNCLASSIFIED IDA/HQ-85-30596 MDA903-84-C-0031 F/G 9/2 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6

Copy 70 of 75 copies

AD-A168 973

IDA PAPER P-1882

C^3 EVAL MODEL DEVELOPMENT AND TEST

Volume II: Programmers Manual

Robert F. Robinson
Joseph W. Stahl
M. L. Roberson
Applications Research Corporation
D. W. Roberson
Applications Research Corporation

October 1985

DTIC FILE COPY

Prepared for
Joint Chiefs of Staff

INSTITUTE FOR DEFENSE ANALYSES
1801 N. Beauregard Street, Alexandria, Virginia 22311

86

IDA Log No. HQ 85-30596

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY
~~DD Form [illegible] dated 1 October [illegible]~~

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for public release; distribution unlimited

4. PERFORMING ORGANIZATION REPORT NUMBER (S)
IDA Paper P-1882

5. MONITORING ORGANIZATION REPORT NUMBER (S)

6a. NAME OF PERFORMING ORGANIZATION
Institute for Defense Analyses

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
OUSDRE (DoD-IDA Management Office)

6c. ADDRESS (City, State, and Zip Code)
1801 N. Beauregard Street
Alexandria, VA 22311

7b. ADDRESS (CITY, STATE, ANMD ZIP CODE)
1801 North Beauregard Street
Alexandria, VA 22311

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
Joint Chiefs of Staff (J-5)

8b. OFFICE SYMBOL (If applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
MDA 903 84 C 0031

8c. ADDRESS (City, State, and Zip Code
The Pentagon
Washington, D.C. 20301-5000

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT	PROJECT NO.	TASK NO.	ACCESSION NO. WORK UNIT
		T-5-309	

11. TITLE (Include Security Classification)
C^3 EVAL MODEL DEVELOPMENT AND TEST - Volume II - Programmers Manual

12. PERSONAL AUTHOR (S).
Robert F. Robinson, Joseph W. Stahl, M.L. Roberson (ARC), D.W. Roberson (ARC)

13. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
FINAL	FROM ____ TO ____	1986, October	222

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Command, Control, communications comba assessment, methodology, simulation, games, analysis, effectiveness, measures

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This is an interim report on the extension and development of the C^3 EVAL model. The model is to permit assessment of the effects on combat of changes in command and control processes and communications network structure and capacity. The model has had a partial pre-processor added to assist possible analyst/model users to input data and a post-processor to provide graphic display of some outputs. The command structure includes the Central European command nodes from division to SHAPE for U.S. forces. The nodes have had input and output limits added to permit representation of degraded operation as under attack or when the unit is moving. The corps level force allocation procedure has been improved. Some processes have been randomized. The corps operates on information different from that available at the division or combat unit due to time delays, randomnbess, and scenario inputs. The impact of changes in the C^3 system can be seen in changes in weapon losses, non-arrival of close air support, and messages delays/lost as well as other operations related elements.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION

22a. NAME OF RESPONSIBLE INDIVIDUAL

22b. TELEPHONE (Include Area Code)

22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

IDA PAPER P-1882

C^3 EVAL MODEL DEVELOPMENT AND TEST

Volume II: Programmers Manual

Robert F. Robinson
Joseph W. Stahl
M. L. Roberson
Applications Research Corporation
D. W. Roberson
Applications Research Corporation

October 1985

IDA

INSTITUTE FOR DEFENSE ANALYSES

Contract MDA 903 84 C 0031
Task T-5-309

PREFACE

This effort was undertaken in March 1985 as a part of a continuing program to develop the C3EVAL model as an analytic tool for use by the Office of the Joint Chiefs of Staff/Command Control and Communications Systems (OJCS/C3S) under Contract No. MDA 903-84C-0031, Task Order No. T-5-309. The basic model has been developed by IDA with programming support from Applications Research Corporation (ARC). This work is reported in IDA Paper P-1756, "Development of C3 Assessment Methodology: The C3EVAL Model," dtd February 1984. This is a report on work in progress and provides a description of the work done in FY1985, an update of the users' manual, and a briefing on the model and its current capabilities.

CONTENTS

FIGURES

C^3EVAL PROGRAMMERS' MANUAL

A. INTRODUCTION

This manual is to be used in conjuction with an Analysts' Manual for the C^3EVAL model. This manual describes the C^3EVAL command, control and communications model and its preprocessor and post processor The preprocessor consists of 37 subroutines of approximately 2,378 lines of code. The post processor consists of 18 subroutines of approximately 936 lines of code. The C^3EVAL (85) model has 70 subroutines and approximately 6,314 lines of code. The source code for the model is written in FORTRAN for a VAX 11/785. Each of the program structures (preproc, C^3EVAL, postproc) are presented in separate sections. There is some duplication of subroutine names in preproc and C^3EVAL. In those cases where the subroutine code is actually used by both programs, it is noted in the preproc description. In each section the data structures, notes on the program and extracts of the comments that are in the code as internal documentation. Computer listings of the codes are in the attachments.

The preprocessor utilizes the DEC Forms Management System (FMS) to communicate with the users. The post processor is based on DECGRAPH for continuous and bar graph output. The C^3EVAL requires the normal FORTRAN library routines including user options to use the random number generator.

B. Program Structure

The source code and development data files are contained in the DRA: [C^3EVAL.UNCLAS] directory on the IDA VAX 11/785 computer under user identification code CAG - 060107. Some model

facilities are under continuing development. Those functions that are in this status (i.e., preproc Limits, CommNet,..and helicopter allocation in C^3EVAL) are identified in the applicable sections. Figure 1 shows the functional and file relationships between programs. The C^3EVAL input files can be modified by use of a general purpose editor and contain data preambles and comment areas to assist a user in this mode. The preproc work file is binary and is not useful to a general purpose editor. All files shown may be saved for future reference and comparison of results.

B.1. Program Preproc

The preprocessor was written in order to facilitate the creation and modification of the data base required to run C3EVAL. The program is a mixture of FORTRAN subroutines and FMS commands for communication with the user. The data base for C3EVAL is 9 data sets contained in 4 input files. The elements of data set are inter-referencing during validation. The preprocessor also allows the user to use names to identify data base. The preprocessor is menu driven with scrolling in fields where it is required. This simplifies what the user needs to know in order to create the data base because the user does not need to know all the in's and out's of an editor. Whenever an invalid input is received an error message is flashed at the bottom of the screen. Figure 2 presents the main menu for the preprocessor as shown on the terminal. When the user indicates the EXIT function output file dispositions are queried by the prepro-cessor.

B.1a. Exit

This option allows the user to leave the main menu and return to the main program. The main program can then save the

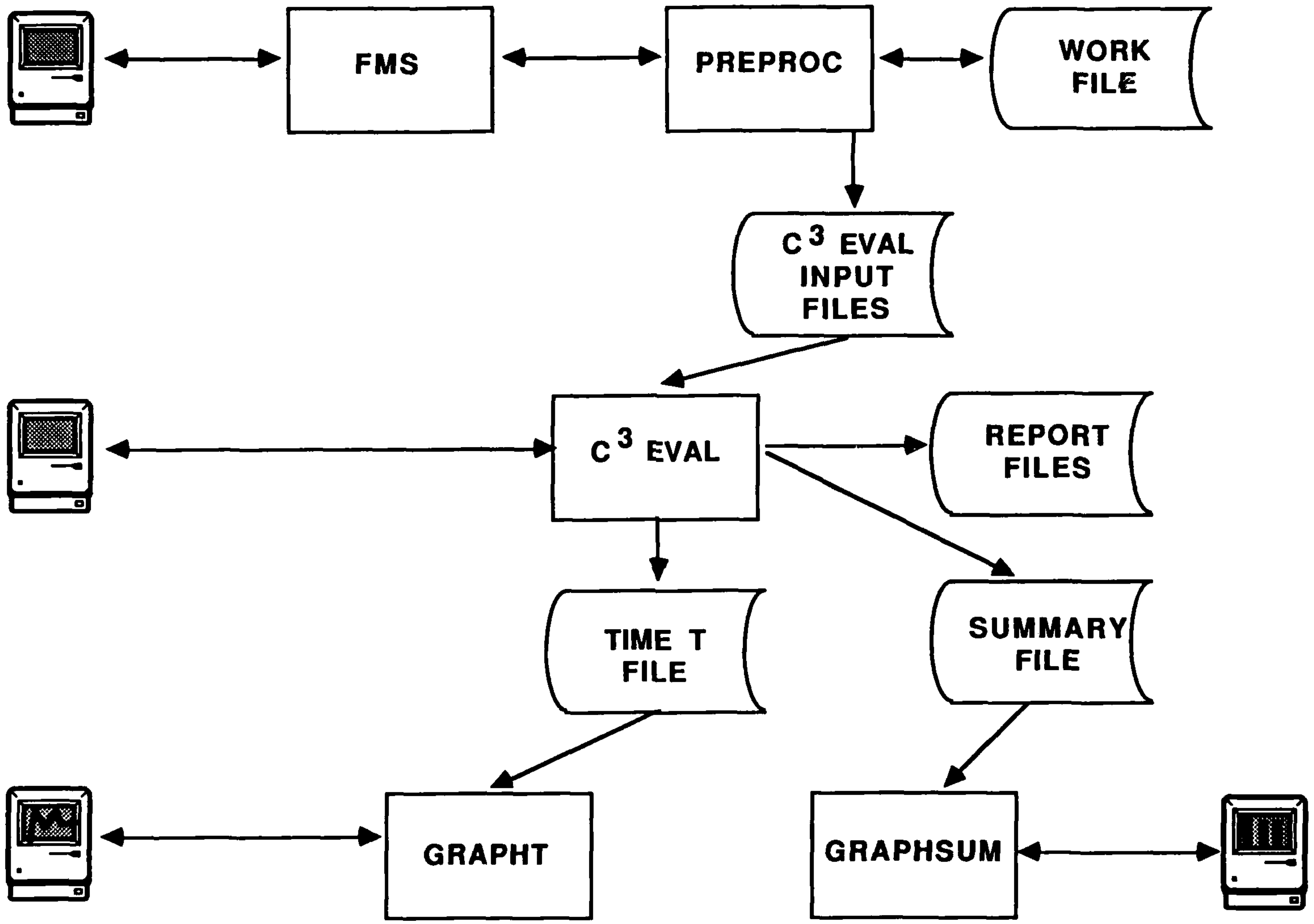


Figure 1. C^3Eval Program Functions

10-17-85-11M
Unclassified

```
  1.  EXIT                                    1.  CREATE
  2.  INSTRUCTIONS                            2.  EDIT
  3.  PREAMBLE DOCUMENTATION
  4.  SIMULATION CONTROL                      SELECT MODE (1-2):  1
  5.  NODE DICTIONARY
  6.  NODE
  7.  LIMITS
  8.  COMMUNICATIONS NETWORKS
  9.  EXTERNAL MESSAGES
 10.  COMBAT DATA
 11.  AIRCRAFT DATA
 12.  HELICOPTER DATA
 13.  RULES

SELECT OPTION NUMBER (1-12):  6
```

Figure 2.

contents of virtual memory or create the data file if the user indicates that he wants either file.

B.1b. Subroutine Instruc

Subroutine Instruc puts the instruction form on the screen and waits for the user to hit the < RETURN > key. The instruction form contains information on how to move the cursor around the screen and other special function eys. The special functions include: browsing up or down a queue, searching for an entry in a queue and deleting entries from a queue. Figure 3 shows the instruction screen.

B.1c. Subroutine Pream

This subroutine allows the user to make preamble documentation for the beginning of the data file. Each line of documentation is 60 characters long. Lines can be changed or added to the bottom but not deleted or inserted.

```
                         INSTRUCTION SCREEN

MOVING FROM ONE FIELD TO ANOTHER:
  NEXT FIELD     = < TAB >
  PREVIOUS FIELD = < BACKSPACE >

SCROLLING IN A SCROLLED AREA:
  SCROLL NEXT     = DOWN_ARROW
  SCROLL PREVIOUS = UP_ARROW
  EXIT SCROLLED AREA NEXT     = < PF1 >
  EXIT SCROLLED AREA PREVIOUS = < PFI >

SEARCHING FOR ENTRY = KEYPAD 4
  NOTE:  WHEN RESPONDING TO PROMPT HIT < ENTER > NOT < RETURN >

DELETING ENTRY = KEYPAD 4
  NOTE:  WHEN RESPONDING TO PROMPT HIT < ENTER > NOT < RETURN >

LOOKING AT ENTRIES:
  NEXT SCREEN     = KEYPAD 2
  PREVIOUS SCREEN = KEYPAD 8

                                      HIT < RETURN > TO MAIN MENU
```

Figure 3.

B.1d. Subroutine SimCtrl

Subroutine SimCtrl allows the user to set the values of C^3EVAL print control flags, optional output modifier times, debug output flag and debug output start and stop times. For all flags a value of zero represents OFF and a value of 1 represents ON. No other values are accepted as input for flags. All output times must be between 0 and 9999.

B.1e. Subroutine NodeDic

Subroutine NodeDic allows the user to create and edit entries within the node dictionary. Create mode allows the user to create new types and names to correspond to them. Edit mode allows the user to change types, i.e., change all entries of type '100' to type '200'. The user can also change or add names. Both modes allow the user to delete all entries of the current

type. Only edit mode allows the user to search for a particular type. Figure 4 is an example showing that type '300' is a division and it has acceptable abbreviations of 'div' or 'DIV'.

```
TYPE                    NAME
300                   div
                      DIVISION
                      DIV



MODE:  EDIT

                         HIT < RETURN > TO RETURN TO MAIN MENU
```

Figure 4: DATA DICTIONARY SCREEN

B.1.e(1) Subroutine Browse

Used for sequentially searching through the node dictionary while in edit mode. The current contents of the screen is all dictionary entries corresponding to the current type number. If the user hits the browse up key then the subroutine gets the locations of all entries corresponding to the previous type. If no previous type exists then sends 'Top of Queue' message to user and keeps the pointers to the current dictionary entries. If the user hits the browse down key then the subroutine gets the locations of all entries corresponding to the next type. If no next type exists then sends 'Bottom of Queue' message to user and keeps the pointers to the current dictionary entries.

B.1.e(2) Subroutine DelDic

Used for deleting all entries in the node dictionary corresponding to the current type, i.e., all entries that are on

the screen when the user hits the appropriate key. There are two pointers, TOP and BOTTOM, which point to the first and last entries of the current type. Starts at location TOP and walks through the queue using the pointers which are sorted by type. Each entry encountered is removed from both dictionary queues and its virtual memory space is released for future use. Stops when it encounters location BOTTOM.

B.1.e(3). Subroutine FindDic

This subroutine is called when the user hits the find key while editing the node dictionary. The type to search for is input by the user. Then subroutine Find is called to get the pointer to the first occurrence of the input type within the node dictionary. The pointer returned by Find is stored in TOP. Since all entries of the same type are grouped together the dictionary is walked through starting at location TOP until the last entry of the input type is found. The pointer to the last entry is stored in BOTTOM. If the type to search for is not found in the node dictionary (find returns a zero) then get first type that is in node dictionary (set TOP to zero).

B.1.e(4). Subroutine GetNew

Subroutine GetNew gets from the user an entry that is not an already existing entry. The field to input from and the queue to search are both parameters. The offset from the beginning of a queue element to compare on is also a parameter. If the user input entry is found in the specified queue then an error message is sent and the user must input again. If the user input entry is not found in the specified queue then the input is passed back to the calling routine.

B.1.f. Subroutine Node

Subroutine Node allows the user to create and edit the node data set. In create mode the user creates new nodes and gives information about the node. This information includes the main node's commander, the main node's subordinates and any other nodes that the main node can communicate with. There can also be two alternate communication nodes for each node that the main node can communicate with. In edit mode the user is allowed to change information about existing nodes. Both modes allow the user to delete the current node from the node queue. Only edit mode allows the user to search for a specific node or to browse up or down the node queue.

B.1.f(1). Subroutine DelNode

Subroutine DelNode deletes a node from the node queue. When a node is deleted its name is set to "DELETED" and it is resorted into the node queue. The node is not removed entirely from the node queue because all other nodes that have the deleted node listed as a commander, subordinate, etc. would either find the wrong node or garbage when it accessed the pointer to the deleted node. When the node is deleted the pointers to the commander and its alternates are set to zero. All entries in the subordinate and other network node queues are removed from the queues and their virtual memory space returned for future use.

B.1.f(2). Subroutine FindNode

This subroutine is called when the user hits the find key while editing a node. The user inputs the name of the node to search for. Then subroutine Find is used to get the pointer to the entry in the node queue that has the input name. If the node name is found in the node queue then the current position is set to the pointer to the entry. If the node name is not found (find

returns a zero) then the current position is set to zero and a message is sent to the user.

B.1.f(3). Subroutine GetName

Subroutine GetName is used to get a valid node name from the user. If the input name is an already existing node then the pointer to that node is returned. Otherwise, if the name contains an entry in the node dictionary then a new node is created with its name being the input name. This new node is sorted into the node queue and the pointer to its location is the value returned by GetName. If the input name is neither an already existing node nor a node name that contains an entry in the node dictionary then the input name is illegal. The subroutine sends an error message to the user and gets another node name from the user.

B.1.f(4). Subroutine GetNew

(See B.1.e. (4))

B.1.f(5). Subroutine GetScr

Subroutine GetScr gets from the user a sequence of valid node names from a scrolled area. A valid node name is either a node that is already in the queue or a node name that contains a word that is in the node dictionary. For each node name in the sequence, if the node name already exists then save the pointer to its location in virtual memory. Otherwise, if the node contains a word that is in the dictionary then create a new entry for the queue and save its pointer. Otherwise, the node name is illegal and the user must input another node name. The values returned are the 3 pointers to the nodes input by the user.

B.1.f(6). Subroutine ScrBk

Subroutine ScrBk is called whenever the user hits the up_arrow key while in a scrolled area. If the current line of the scrolled area is not the top line then the new current line becomes the line above the current line. If the current line of the scrolled area is the top line and there are undisplayed lines above the current line then each line of the scrolled area is moved down and the new line is displayed at the top of the scrolled area.

B.1.f(7). Subroutine ScrFwd

Subroutine ScrFwd is called whenever the user hits the down_arrow key while in a scrolled area. If the current line of the scrolled area is not the bottom line then the new current line becomes the line below the current line. If the current line of the scrolled area is the bottom line and there are undisplayed lines below the current line then each line of the scrolled area is moved up and the new line is displayed at the bottom of the scrolled area.

B.1.g. Subroutine Limits

Subroutine Limits is not implemented yet.

B.1.h. Subroutine CommNet

Subroutine CommNet is not implemented yet.

B.1.i. Subroutine ExtMsg

Subroutine ExtMsg is not implemented yet.

B.1.j. Subroutine CbData

Subroutine CbData is not implemented yet.

B.1.k. Subroutine AcData

Subroutine AcData is not implemented yet.

B.1.l. Subroutine HcData

Subroutine hcData is not implemented yet.

B.1.m. Subroutine Rules

Subroutine Rules is not implemented yet.

B.1.n. Utilities

B.1.n(1). Subroutine DMInit

Same as subroutine DMInit in program C3EVAL. See section B.2.a(2)

B.1.n(2). Subroutine Find

Same as subroutine Find in program C3EVAL. See section B.2.e(1)

B.1.n(3). Subroutine GetTyp

Subroutine GetTyp searches a node name for any word that occurs in the node dictionary. If an occurrence of a word in node name is found then returns the type corresponding to the dictionary entry. Otherwise, returns a string of blanks.

B.1.n.(4). Subroutine GetWord

Subroutine GetWord finds the first word that is contained in a string. If the string passed in is blank then GetWord returns blanks for the string and the word. Otherwise, the first word within the string is found and saved in IWORD. Then the word is

removed from the string and GetWord returns the resulting string and IWORD.

B.1.n(5). Subroutine Gimme

Same as subroutine Gimme in program C^3EVAL. See section B.2.e(2).

B.1.n(6). Subroutine IntChr

Subroutine IntChr converts an integer to its ASCII representation. The parameter ISize is the number of digits to convert. Note that the maximum length of the string is 12 characters by declaration.

B.1.n(7). Subroutine POut

Same as subroutine POut in program C^3EVAL. See section B.2.e(3).

B.1.n(8). Subroutine Release

Same as subroutine Release in program C^3EVAL. See section B.2.e(4).

B.1.n(9). Subroutine Restore

Same as subroutine Restore in program C^3EVAL. See section B.2.e(5).

B.1.n(10). Subroutine Save

Same as subroutine Save in program C^3EVAL. See section B.2.e(6).

B.1.n(11). Subroutine ScrLine

Subroutine ScrLine is used to create a string to output to a scrolled line. The scrolled line contains 6 fields consisting of: main node name, main node id, 1st alternate's name, 1st alternate's id, 2nd alternate's name and 2nd alternate's id. Each name is a string of 12 characters and each id is a string of 4 characters. Each name and id is found by using the appropriate pointer to get the node's location in memory and then picking up the name and id from the appropriate offsets. When the resulting string is passed to FMS it will be parsed and each value will be sent to the corresponding field.

B.1.n(12). Subroutine SetFlag

Subroutine SetFlag creates the data structure that contains the print control flags, optional print modifiers, debug print flag and debug print start and stop times. All values are initialized to zero. Since all flags are only 1 character but are stored in 4 character fields the 1st character of the field is initialized to zero. However, since the optional print modifiers and debug print start and stop times are 4 character modifiers and debug print start and stop times are 4 character fields that are right justified the zero is in the last location.

B.1.n(13). Subroutine SnapQ

Subroutine SnapQ inserts an entire queue of records into another queue of records. Assumes that all records in the queue being added have the same value being sorted on, therefore, they can be inserted as one large record. Note: A queue of one record can be inserted by passing the same pointer for both the top and bottom of the queue to be added. Assumes that there is a corresponding back pointer for the forward pointer. Assumes back pointer is offset from its forward pointer by 1.

B.1.n(14). Subroutine UnSnap

Subroutine UnSnap removes an entry from a queue. Assumes that there is a corresponding back pointer for each forward pointer. Assumes back pointer is offset from its forward pointer by 1. Sets forward pointer of previous node to next node. Sets back pointer of next node to previous node.

B.1.n(15). Function Validl

Function Validl is a Field Completion User Action Routine. Validl checks to see that the inputted value is between 1 and a maximum value. The maximum value is dependent on the field that is being read from. The maximum value is stored in Named Data which is an FMS data structure.

B.1.0. Data Structures

BLOCK NAME:	MROOT		BLOCK SIZE: 30
USE:	Contains all root pointers for virtual memory		
CREATED BY:	PREPROC		
DELETED BY:	not applicable		
ROOT:	COMMON/LOCATE/MROOT		DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PDICNI	P	NODE dictionary (sorted by TYPE)
2	PDICN2	P	NODE dictionary (sorted by NAME)
3	PNODE	P	NODE queue
4	PREAMB	P	Preamble documentation
5	PFLAG	P	Print control flags

BLOCK NAME: **PDICN** BLOCK SIZE: 6

USE: **Dictionary of all valid node types. Each type number has one or more unit names that correspond to that type.**

CREATED BY: **NODEDIC**

DELETED BY: **DELDIC**

ROOT: **MROOT+0** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PDICN1	P	Next dictionary entry (by TYPE)
2	PPREV1	P	Previous dictionary entry (by TYPE)
3	PDICN2	P	Next dictionary entry (by NAME)
4	PPREV2	P	Previous dictionary entry (by NAME)
5	NAME	P	Dictionary entry
6	TYPE	P	Unit type of dictionary entry

BLOCK NAME: **PNODE** BLOCK SIZE: **10**

USE: **Queue containing all nodes for the scenario. Each entry in the queue also has all communication paths that pertain to the node.**

CREATED BY: **GETNAME, GETSCR, NODE**

DELETED BY: **DELNODE**

ROOT: **MROOT+2** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNIDEF	P	Next NODE
2	PNODEB	P	Previous NODE
3	NAME	C	NODE name
4	UNIT	I	NODE number
5	TYPE	C	NODE type
6	PCMDR	P	NODE's commander
7	PCMDR1	P	1st alternate for commander
8	PCMDR2	P	2nd alternate for commander
9	PSUBQ	P	NODE's subordinate queue
10	PCOMQ	P	NODE's network queue

BLOCK NAME: **PSUBQ** BLOCK SIZE: 5

USE: **Queue of all subordinate communications paths for a specified node. Each entry in the queue can also have the two alternate communications paths for the subordinate.**

CREATED BY: **NODE**

DELETED BY: **DELNODE**

ROOT: **PNODE+8** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PSUBQF	P	Next subordinate
2	PSUBQB	P	Previous subordinate
3	PSUB	P	NODE's subordinate
4	PSUB1	P	1st alternate for subordinate
5	PSUB2	P	2nd alternate for subordinate

BLOCK NAME: PCOMQ BLOCK SIZE: 5

USE: **Queue of all other communications paths for a specified node. Each entry in the queue can also have the two alternate communications paths.**

CREATED BY: **NODE**
DELETED BY: **DELNODE**
ROOT: **PNODE+9** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PCOMQF	P	Next network NODE
2	PCOMQB	P	Previous network NODE
3	PCOM	P	Network NODE
4	PCOM1	P	1st alternate for network NODE
5	PCOM2	P	2nd alternate for network NODE

BLOCK NAME: **PREAMB** BLOCK SIZE: 3

USE: **Linked list of preamble documentation lines. Each line is 60 characters long.**

CREATED BY: **PREAM**

DELETED BY: **not applicable**

ROOT: **MROOT+3** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PREAMBF	P	Pointer to next line
2	PREAMBB	P	Pointer to previous line
3	LINE	C*60	Line of documentation

BLOCK NAME: PFLAG — BLOCK SIZE: 26

USE: **List of all print control flags, optional output modifier times, debug output flag and debug output start and stop times. NOTE (for all flags): 0 -- > OFF, 1 -- > ON.**

CREATED BY: **SETFLAG**

DELETED BY: **not applicable**

ROOT: **MROOT+4** — DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	FLAG1	C*1	All messages at alternate dest.
2	FLAG2	C*1	All messages on input queues
3	FLAG3	C*1	All messages on output queues
4	FLAG4	C*1	All messages on future queues
5	FLAG5	C*1	All messages being held
6	FLAG6	C*1	All messages being deleted
7	FLAG7	C*1	Status of rule structure
8	FLAG8	C*1	CAS take off scheduled
9	FLAG9	C*1	not assigned
10	FLAG10	C*1	not assigned
11	FLAG11	C*1	Tracked messages at alternate dest.
12	FLAG12	C*1	Tracked messages on input queues
13	FLAG13	C*1	Tracked messages on out queues
14	FLAG14	C*1	Time T output on file 14 required
15	FLAG15	C*1	Combat loss vector
16	FLAG16	C*1	Force ratio calculations
17	FLAG17	C*1	Rule status at final time
18	FLAG18	C*1	not assigned
19	FLAG19	C*1	Random processing required
20	FLAG20	C*1	Used internally for sum of flags
21	MOD1	C*4	Optional output restricted to this node
22	MOD2	C*4	Optional output starts at this time
23	MOD3	C*4	Optional output stops at this time
24	DEBUG1	C*1	Debug output flag
25	DEBUG2	C*4	Debug output start time
26	DEBUG3	C*4	Debug output stop time

B.1.p. Program Notes

Most of the interfacing with the screen is accomplished by using FMS provided routines and structures. One tool provided by FMS is a Field Completion User Action Routine. A field completion UAR is a function that is called by FMS when the user completes his entry for a field. The function can then process the value that the user input to determine if it is a legal entry. The function returns a value which tells FMS to either accept the user input or to get another value from the user. Named Data is another tool provided by FMS. These are data values that have names so that they can be accessed by an FMS command. This way values can be associated with particular fields but not hard-wired into the actual code. More detailed information on UAR's, Named Data, or any of the FMS provided routines can be obtained from VAX manuals on FMS.

B.1.q. Internal Code Documentation

```
PROGRAM PREPROC

PURPOSE:  CREATE DATA FILE TO BE USED AS INPUT FOR
   PROGRAM C3EVAL

INITIALIZE FMS
IF WORKFILE EXISTS LOAD DYNAMIC MEMORY ELSE INITIALIZE
 DYNAMIC MEMORY
PROCESS ALL MENU REQUESTS
SAVE CONTENTS OF MEMORY AND COMMON'S
CREATE OUTPUT FILE
CLEAN UP
1 SUBROUTINE ACDATA
SUBROUTINE IS NOT IMPLEMENTED YET
1 SUBROUTINE BROWSE(NWORD, FLAG)

PURPOSE:  USED WHEN EDITING NODE DICTIONARY. MOVES UP OR DOWN ONE
  SCREEN, I.E. GETS ENTRIES CORRESPONDING TO PREVIOUS OR NEXT TY

PARAMETERS:
   NWORD    ==> OFFSET TO COMPARE ON
   FLAG     ==> DIRECTIONAL FLAG
                0 -> SEARCH DOWN
                1 -> SEARCH UP

IF NO NEXT ENTRY THEN SEND APPROPRIATE PROMPT
SET TOP TO NEXT ENTRY
FIND ALL ENTRIES OF SAME TYPE AS TOP
SET BOTTOM TO LAST ENTRY OF SAME TYPE AS TOP

IF SEARCHING UP THEN SWITCH TOP AND BOTTOM POINTERS AND
 SUBTRACT ONE TO GET FORWARD POINTERS INSTEAD OF PREVIOUS
 POINTERS.

1 SUBROUTINE CBDATA
SUBROUTINE IS NOT IMPLEMENTED YET
1 SUBROUTINE COMMNET
SUBROUTINE IS NOT IMPLEMENTED YET
1 SUBROUTINE DELDIC

PURPOSE:  DELETE ALL ENTRIES IN NODE DICTIONARY CORRESPONDING TO
  CURRENT TYPE, I.E. ALL ENTRIES ON SCREEN.

TOP AND BOTTOM ARE POINTERS TO THE FIRST AND LAST ENTRIES
 OF THE CURRENT TYPE.
DO FOR ALL ENTRIES BETWEEN TOP AND BOTTOM
UNSNAP CURRENT ENTRY FROM DICTIONARY QUEUE BY TYPE
UNSNAP CURRENT ENTRY FROM DICTIONARY QUEUE BY NAME
RETURN VIRTUAL MEMORY SPACE FOR FURTHER USE
1 SUBROUTINE DELNODE(PNODE)

PURPOSE:   DELETE A NODE FROM THE NODE QUEUE

PARAMETERS:
   PNODE    ==> POINTER TO NODE TO DELETE

SET NODE TYPE TO BLANKS
REMOVE POINTERS TO COMMANDER AND IT'S ALTERNATES
REMOVE ALL ENTRIES FROM NODE'S SUBORDINATE QUEUE AND RELEASE
```

```
 MEMORY S'ACE FOR FUTURE USE.
REMOVE ALL ENTRIES FROM NODE'S NETWORK QUEUE AND RELEASE
 MEMORY SPACE FOR FUTURE USE.
REMOVE MAIN NODE FROM QUEUE; MARK MAIN NODE AS DELETED;
 RESORT MAIN NODE INTO QUEUE
SUBROUTINE DMINIT(MEMORY,MPTR,IGBPTR,MAXDM)
*************************************
*   INITIALIZE DYNAMIC MEMORY
*************************************
INITIALIZE MEMORY POINTER
INITIALIZE GARBAGE POINTER
CLEAR DYNAMIC MEMORY
SUBROUTINE EXTMSG
SUBROUTINE IS NOT IMPLEMENTED YET
SUBROUTINE FILEOUT

PURPOSE:   CREATE OUTPUT FILE FOR FURTHER USE BY PROGRAM C3EVAL

OPEN OUTPUT FILE
OUTPUT PREAMBLE DOCUMENTATION
DO FOR ALL ENTRIES IN QUEUE
OUTPUT CURRENT LINE
OUTPUT LINE THAT SIGNALS END OF PREAMBLE DOCUMENTATION
OUTPUT HEADER LINE FOR PRINT FLAGS
OUTPUT PRINT CONTROL FLAGS, OPTIONAL OUTPUT TIMES, AND DEBUG
 OUTPUT FLAG
OUTPUT DEBUG START AND STOP TIMES
WRITE INPUT MODE AND HEADER LINE
DO FOR ALL ENTRIES IN NODE QUEUE
GET NODE'S UNIT NUMBER AND TYPE
GET UNIT NUMBER OF COMMANDER
GET UNIT NUMBER OF 1ST ALTERNATE FOR COMMANDER
GET UNIT NUMBER OF 2ND ALTERNATE FOR COMMANDER
SET COMMANDER AND SUBORDINATE FLAGS, APPROPRIATELY
OUTPUT NODES COMMANDER AND ITS ALTERNATES
SET COMMANDER AND SUBORDINATE FLAGS, APPROPRIATELY
DO FOR ALL ENTRIES IN SUBORDINATE QUEUE
GET UNIT NUMBER OF SUBORDINATE
GET UNIT NUMBER OF 1ST ALTERNATE FOR SUBORDINATE
GET UNIT NUMBER OF 2ND ALTERNATE FOR SUBORDINATE
OUTPUT NODES SUBORDINATE AND IT'S ALTERNATES
SET COMMANDER AND SUBORDINATE FLAGS, APPROPRIATELY
DO FOR ALL ENTRIES IN NETWORK QUEUE
GET UNIT NUMBER OF NETWORK NODE
GET UNIT NUMBER OF 1ST ALTERNATE FOR NETWORK NODE
GET UNIT NUMBER OF 2ND ALTERNATE FOR NETWORK NODE
OUTPUT NODES NETWORK NODE AND IT'S ALTERNATES
CLOSE OUTPUT FILE
SUBROUTINE FIND(PIN, N, ID, POUT)
**************************************************
*   FIND A POINTER IN A QUEUE
**************************************************
*   INPUT
*      PIN - POINTER TO TOP OF QUEUE TO BE SEARCHED
*      N   - OFFSET FROM PIN TO COMPARE
*      ID  - VALUE TO MATCH WITH N
**************************************************
*   CREATES
*      POUT - POINTER TO DESIRED ELEMENT
```

```
*******************************************************
DO FOR ALL QUEUE ELEMENTS
   COMPARE VALUES
GET NEXT ELEMENT
END OF SEARCH
SUBROUTINE FINDDIC

PURPOSE:  FIND ALL ENTRIES IN NODE DICTIONARY CORRESPONDING
  TO INPUTTED TYPE.

GETS A STRING FROM THE USER CORRESPONDING TO THE TYPE NUMBER
 TO SEARCH FOR
FINDS FIRST ENTRY OF INPUTTED TYPE AND STORES POINTER IN TOP
FIND ALL DICTIONARY ENTRIES OF SAME TYPE
STORE POINTER TO LAST ENTRY OF INPUTTED TYPE IN BOTTOM
SUBROUTINE FINDNODE

PURPOSE:   SEARCH NODE QUEUE FOR USER INPUTTED NODE.

GET NODE NAME TO SEARCH FOR
SEARCH FOR NODE NAME

IF NODE NAME IS NOT IN NODE QUEU THEN ERROR ELSE SET CURRENT
 POSITION POINTER TO NODE JUST FOUND

SUBROUTINE GETNAME(FLDNAM, FLDIDX, FLDTRM, PTEMP)

PURPOSE:  GETS FROM THE USER A VALID NODE NAME.

PARAMETERS:
   FLDNAM --> NAME OF FIELD TO INPUT FROM
   FLDIDX --> INDEX OF FIELD
   FLDTRM --> VALUE OF FIELD TERMINATOR KEY
   PTEMP  --> POINTER TO NODE

GET NODE NAME FROM USER. IF NULL ENTRY THEN RETURN NILL POINTER.
NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
OUTPUT NODE'S UNIT NUMBER TO APPROPRIATE ID FIELD
SUBROUTINE GETNEW(PROOT,LENGTH,FLDNAM,FLDIDX,FLDVAL,FLDTRM)

PURPOSE:  GET UNIQUE ENTRY FROM USER.

PARAMETERS:
   PROOT   --> ROOT OF QUEUE TO GET UNIQUE ENTRY FOR
   LENGTH  --> OFFSET FROM PROOT TO COMPARE
   FLDNAM  --> NAME OF FIELD TO INPUT FROM
   FLDIDX  --> INDEX OF FIELD TO INPUT FROM
   FLDVAL  --> VALUE INPUTTED FROM FIELD
   FLDTRM  --> FMS VALUE OF TERMINATOR KEY

GET ENTRY FROM USER.  IF FIELD INDEX IS 0 THEN DON'T PASS
 TO FMS ROUTINE GET.
SEARCH QUEUE FOR ENTRY
SUBROUTINE GETSCR(FLDNAM,FVALUE,FLDTRM,PTEMP,PTEMP1,PTEMP2)

PURPOSE:  GETS FROM THE USER A SEQUENCE OF VALID NODE NAMES
```

FROM A SCROLLED AREA.

PARAMETERS:
FLDNAM ==> NAME OF FIELD TO INPUT FROM
FVALUE ==> VALUES INPUTTED INTO FIELDS
FLDTRM ==> VALUE OF FIELD TERMINATOR KEY
PTEMP ==> POINTER TO NODE
PTEMP1 ==> POINTER TO 1ST ALTERNATE NODE
PTEMP2 ==> POINTER TO 2ND ALTERNATE NODE

INITIALIZE POINTERS TO ZERO

IF ALL FIELDS ARE BLANK THEN RETURN NILL POINTER.
IF MAIN FIELD IS BLANK BUT ALTERNATE FIELD IS NON-BLANK
THEN SEND ERROR MESSAGE AND GET INPUT AGAIN.

PROCESS NODE NAME

NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
SET UNIT NUMBER OF MAIN NODE

IF THERE IS A 1ST ALTERNATE THEN PROCESS 1ST ALTERNATE.
IF NO 1ST ALTERNATE AND NO 2ND ALTERNATE THEN BRANCH TO
OUTPUT SECTION.
IF NO 1ST ALTERNATE BUT 2ND ALTERNATE THEN ERROR - GET
INPUT AGAIN.

PROCESS 1ST ALTERNATE

NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
SET UNIT NUMBER OF 1ST ALTERNATE
IF THERE IS A 2ND ALTERNATE THEN PROCESS 2ND ALTERNATE
ELSE BRANCH TO OUTPUT SECTION

PROCESS 2ND ALTERNATE

NODE DOESN'T EXIST, THEREFORE, CREATE NEW NODE
ILLEGAL NAME: TRY AGAIN
LEGAL NAME: SAVE NAME, NUMBER AND TYPE
SNAP INTO QUEUE
SET UNIT NUMBER OF 2ND ALTERNATE
OUTPUT LINE TO APPROPRIATE SCROLLED AREA
SUBROUTINE GETTYP(FLDVAL,ITYPE)

PURPOSE: SEARCHES NODE NAME FOR ANY WORD THAT OCCURS IN NODE
DICTIONARY.

PARAMETERS:
FLDVAL ==> NODE NAME
ITYPE ==> UNIT TYPE OF FLDVAL

```
        DO FOR EACH WORD IN NODE NAME
        GET NEXT WORD FROM STRING. SEARCH DICTIONARY.
        FOUND WORD IN DICTIONARY; SAVE TYPE
        DID NOT FIND WORD IN DICTIONARY; SET TYPE TO BLANKS
1       SUBROUTINE GETWORD(FVALUE, IWORD)

        PURPOSE:  FINDS THE FIRST WORD CONTAINED IN A STRING.

        PARAMETERS:
           (INPUT)
           FVALUE  ==> STRING CONTAINING WORDS
           (OUTPUT)
           FVALUE  ==> INITIAL VALUE WITH 1ST WORD REMOVED
           IWORD   ==> 1ST WORD CONTAINED IN FVALUE

        IF ALL BLANKS THEN NO WORD IN STRING
        REMOVE LEADING BLANKS
        IF LOCATION OF BLANK IS 0 THEN NO BLANK FOUND AND WORD IS ENTIRE
          STRING.  OTHERWISE, WORD IS ALL LOCATIONS IN STRING PRECEDING
          LOCATION OF BLANK.
        SUBROUTINE GIMME(NPTR, LEN, ISPACE)

        SEGMENT GET VIRTUAL SPACE

        PARAMETERS:
           NPTR    ==> POINTER TO BLOCK ALLOCATED
           LEN     ==> LENGTH OF BLOCK TO ALLOCATE
           ISPACE ==> VIRTUAL MEMORY TO GET BLOCK FROM

        SEARCH GARBAGE LIST
        DO UNTIL LIST ENDS
           IF (SIZE.EQ.LENGTH) THEN
           SET PTR TO FIRST BLOCK
           SNAP GARBAGE PTR
        ALLOCATE VIRGIN STORAGE
        UPDATE VIR SPACE PTR
        STORAGE OVERFLOW
        ZERO SPACE BLOCK
        END SEGMENT
        SUBROUTINE HCDATA
        SUBROUTINE IS NOT IMPLEMENTED YET
1       SUBROUTINE INSTRUC

        PURPOSE:   PUT FORM CONTAINING INSTRUCTIONS ON SCREEN.
        PUT INSTRUCTION FORM ON SCREEN
        WAIT FOR USER TO HIT <RETURN>
1       SUBROUTINE INTCHR(VALUE, ISIZE, STRING)

        PURPOSE:  CONVERTS AN INTEGER TO ITS ASCII REPRESENTATION.

        PARAMETERS:
           VALUE   ==> INTEGER VALUE TO CONVERT
           LENGTH  ==> NUMBER OF DIGITS TO CONVERT
           STRING  ==> ASCII REPRESENTATION OF VALUE

1       SUBROUTINE LIMITS
        SUBROUTINE NOT IMPLEMENTED YET
1       SUBROUTINE MENU
```

PURPOSE: ALLOWS USER TO SELECT WHICH FUNCTIONS ARE IMPLEMENTED IN THE PREPROCESSOR.

PUT MAIN MENU ON SCREEN
GET MODE FROM USER AND CONVERT TO INTEGER
GET OPTION FROM USER AND CONVERT TO INTEGER
SUBROUTINE NODE

PURPOSE: USED TO CREATE AND EDIT THE NODE DATA SET.

FMS TERMINATOR CODES

********** CREATE MODE **********

GET MAIN NODE

NODE NAME MUST BE UNIQUE AND MUST CONTAIN ONE WORD WHICH IS IN THE NODE DICTIONARY.

STORE NAME, NUMBER, AND TYPE
SNAP INTO QUEUE

PROCESS FIELD TERMINATOR

GET COMMANDER

SAVE POINTER TO COMMANDER

PROCESS FIELD TERMINATOR

GET 1ST ALTERNATE FOR COMMANDER

IF NO COMMANDER THEN SEND MESSAGE; REMOVE 1ST ALTERNATE FROM SCREEN; BRANCH TO GET COMMANDER
SAVE POINTER TO 1ST ALTERNATE

PROCESS FIELD TERMINATOR

GET 2ND ALTERNATE FOR COMMANDER

IF NO COMMANDER THEN SEND MESSAGE; REMOVE 2ND ALTERNATE FROM SCREEN; BRANCH TO GET COMMANDER
IF NO 1ST ALTERNATE THEN SEND MESSAGE; REMOVE 2ND ALTERNATE FROM SCREEN; BRANCH TO GET 1ST ALTERNATE
SAVE POINTER TO 2ND ALTERNATE

PROCESS FIELD TERMINATOR

GET SUBORDINATE AND ITS ALTERNATES

GET USER ENTRY
VERIFY ENTRIES

IF PSUB IS 0 THEN NULL ENTRY, THEREFORE, NO MORE SUBORDINATES
ADD SUBORDINATE TO BOTTOM OF QUEUE

PROCESS FIELD TERMINATOR

GET NETWORK NODES

GET USER ENTRY
VERIFY ENTRIES
IF PCOM IS 0 THEN NULL ENTRY, THEREFORE, NO MORE NETWORK NODES
ADD NETWORK NODE TO BOTTOM OF QUEUE

PROCESS FIELD TERMINATOR

********** EDIT MODE **********

PRINT MAIN NODE AND ID
PRINT COMMANDER AND ID
PRINT 1ST ALTERNATE FOR COMMANDER AND ID
PRINT 2ND ALTERNATE FOR COMMANDER AND ID
PRINT SUBORDINATES AND THEIR ALTERNATES AND ID'S
PRINT OTHER NETWORK NODES AND THEIR ALTERNATES AND ID'S

GET NAME AND INDEX OF CURRENT FIELD. SAVE OLD VALUE.

GET ENTRY FROM USER

IF ENTRY EQUALS OLD VALUE THEN NO CHANGE - BRANCH TO PROCESS
FIELD TERMINATOR

CHANGE MAIN NODE NAME

MAIN NODE NAME MUST BE UNIQUE
DETERMINE IF NAME IS VALID, I.E. UNIT WORD OCCURS IN DICTIONARY
LEGAL NAME - SAVE NAME AND TYPE

CHANGE COMMANDER OR IT'S ALTERNATES

LENGTH IS THE OFFSET FROM CURPOS FOR THE POINTER FOR
EITHER THE COMMANDER OR IT'S ALTERNATES

PROCESS BLANK ENTRY

IF ENTRY WAS IN COMMANDER FIELD:
   IF NO ALTERNATES THEN IT IS LEGAL TO DELETE COMMANDER.
   IF ALTERNATES EXIST THEN SEND MESSAGE, PUT OLD VALUE FOR
   COMMANDER BACK ON SCREEN, AND BRANCH TO PROCESS FIELD
   TERMINATOR KEY.
IF ENTRY WAS IN 1ST ALT. FOR COMMANDER FIELD:
   IF NO 2ND ALTERNATE THEN IT IS LEGAL TO DELETE 1ST ALTERNATE.
   IF 2ND ALTERNATE EXISTS THEN SEND MESSAGE, PUT OLD VALUE FOR
   1ST ALT. BACK ON SCREEN, AND BRANCH TO PROCESS FIELD TERMINATOR
   KEY.

PROCESS A NON-BLANK ENTRY

IF ENTRY WAS IN 1ST ALT. FOR COMMANDER FIELD:
   IF COMMANDER EXISTS THEN IT IS LEGAL TO ADD 1ST ALTERNATE.
   IF NO COMMANDER THEN SEND MESSAGE, REMOVE 1ST ALTERNATE FROM SCREEN, AND BRANCH TO PROCESS FIELD TERMINATOR KEY.
IF ENTRY WAS IN 2ND ALT. FOR COMMANDER FIELD:
   IF 1ST ALTERNATE EXISTS THEN IT IS LEGAL TO ADD 2ND ALTERNATE.
   IF NO 1ST ALTERNATE THEN SEND MESSAGE, REMOVE 2ND ALTERNATE FROM SCREEN, AND BRANCH TO PROCESS FIELD TERMINATOR KEY.
DETERMINE IF NAME IS UNIQUE
ENTRY DOESN'T EXIST; CHECK IF VALID NAME, I.E. UNIT WORD OCCURS IN DICTIONARY
CREATE NODE
SAVE POINTER TO NODE
WRITE NEW ID TO SCREEN AND BRANCH TO PROCESS FIELD TERMINATOR KEY

CHANGE SUBORDINATE AND ALTERNATES

GET SUBORDINATE NAME FROM FIELD
GET 1ST ALTERNATE FROM FIELD
GET 2ND ALTERNATE FROM FIELD
VERIFY ENTRIES

ADDING NEW ENTRIES

CREATE NEW ENTRY FOR SUBORDINATE QUEUE
SET BACK POINTER OF NEW ENTRY TO LAST ENTRY OF QUEUE
SET SUBORDINATE AND ALTERNATE POINTERS TO APPROPRIATE VALUES
IF QUEUE IS EMPTY THEN ADD NEW ENTRY TO TOP ELSE ADD TO BOTTOM
BRANCH TO PROCESS FIELD TERMINATOR KEY

CHANGING EXISTING ENTRIES

BRANCH TO PROCESS FIELD TERMINATOR KEY

CHANGE NETWORK NODES AND ALTERNATES

GET NETWORK NAME FROM FIELD
GET 1ST ALTERNATE FROM FIELD
GET 2ND ALTERNATE FROM FIELD
VERIFY ENTRIES

ADDING NEW ENTRIES

CREATE NEW ENTRY FOR NETWORK QUEUE
SET BACK POINTER OF NEW ENTRY TO LAST ENTRY OF QUEUE
SET NETWORK AND ALTERNATE POINTERS TO APPROPRIATE VALUES
IF QUEUE IS EMPTY THEN ADD NEW ENTRY TO TOP ELSE ADD TO BOTTOM
BRANCH TO PROCESS FIELD TERMINATOR KEY

CHANGING EXISTING ENTRIES

PROCESS FIELD TERMINATOR

SUBROUTINE NODEDIC

```
PURPOSE:   CREATE AND EDIT ENTRIES WITHIN THE NODE DICTIONARY.


FMS TERMINATOR CODES

**********  CREATE MODE  **********
GET TYPE
GET NAMES
SORT BY TYPE
SORT BY NAME
**********  EDIT MODE  **********
SET TOP TO FIRST ENTRY IN DICTIONARY. SET BOTTOM TO LAST ENTRY
  THAT HAS SAME TYPE AS FIRST ENTRY.
INITIALIZE FIELDS
CHANGE EXISTING NAME
ADD NEW NAME
SORT BY TYPE
SORT BY NAME
IF NEW NAME WAS ADDED BEFORE FIRST ENTRY OF IT'S TYPE
 THEN UPDATE POINTER TO TOP
IF NEW NAME WAS ADDED AFTER LAST ENTRY OF IT'S TYPE
 THEN UPDATE POINTER TO BOTTOM
CHANGE EXISTING TYPE
RESORT TYPE
PROCESS FIELD TERMINATOR
1   SUBROUTINE POUT(MEMORY, NUM, LENGTH)

PURPOSE:
   PRINT OUT CONTENTS OF VIRTUAL MEMORY

PARAMETERS:
   MEMORY  --› VIRTUAL MEMORY TO PRINT
   NUM     --› NUMBER OF LINES TO PRINT
   LENGTH  --› NUMBER OF VALUES TO PRINT PER LINE

1   SUBROUTINE PREAM

PURPOSE:   ALLOW USER TO CREATE AND EDIT THE PREAMBLE
   DOCUMENTATION.


FMS TERMINATOR KEYS


*** CREATE MODE ***

CLEAR DISPLAY AND PUT THE FORM FOR THE PREAMBLE ON THE SCREEN
INITIALIZE THE CURRENT FIELD NAME AND INDEX
DO UNTIL USER HITS ‹RETURN› KEY
GET USER ENTRY FROM CURRENT FIELD
FIRST ENTRY - SAVE VALUE; SET ROOT POINTER
QUEUE NOT EMPTY - SAVE VALUE; ADD TO BOTTOM OF QUEUE

PROCESS FIELD TERMINATOR

IF ‹TAB› KEY THEN SET CURRENT FIELD TO NEXT FIELD
IF ‹RETURN› KEY THEN BRANCH TO END OF SUBROUTINE

*** EDIT MODE ***
```

```
IF QUEUE IS EMPTY THEN CANNOT EDIT; BRANCH TO END OF
 SUBROUTINE
CLEAR DISPLAY AND PUT THE FORM FOR THE PREAMBLE ON THE SCREEN
INITIALIZE POINTER TO TOP OF QUEUE; INITIALIZE CURRENT FIELD
 NAME AND INDEX
PUT EXISTING DOCUMENTATION ON SCREEN
SET POINTER TO TOP OF QUEUE AND FIELD INDEX TO ONE
DO UNTIL USER HITS ‹RETURN› KEY
GET USER ENTRY FROM CURRENT FIELD
NEW LINE - SAVE VALUE; ADD ENTRY TO BOTTOM OF QUEUE; UPDATE
 POINTER TO LAST LINE; BRANCH TO PROCESS FIELD TERMINATOR
OLD LINE - UPDATE VALUE

PROCESS FIELD TERMINATOR

IF ‹TAB› KEY THEN SET CURRENT FIELD TO NEXT FIELD
IF ‹BACKSPACE› KEY THEN SET CURRENT FIELD TO PREVIOUS FIELD
IF ‹RETURN› KEY THEN BRANCH TO END OF SUBROUTINE
SUBROUTINE QPRINT(PROMPT)

PURPOSE:   PRINT OUT CONTENTS OF NODE DICTIONARY QUEUE

PARAMETERS:
   PROMPT  ==› HEADER STRING TO OUTPUT ALONG WITH CONTENTS
               OF NODE DICTIONARY QUEUE.

OUTPUT HEADER PROMPT
DO FOR ALL ENTRIES IN NODE DICTIONARY QUEUE
OUTPUT FORWARD AND BACKWARD POINTERS FOR BOTH TYPE AND NAME
 SORTING.  OUTPUT TYPE AND NAME.
1 SUBROUTINE RELEAS(NPTR, LEN, ISPACE)

SEGMENT RELEASE PUTS STORAGE ON GARBAGE LIST

PARAMETERS:
   NPTR   ==› POINTER TO BLOCK TO RELEASE
   LEN    ==› LENGTH OF BLOCK TO RELEASE
   ISPACE ==› VIRTUAL MEMORY BLOCK IS CONTAINED IN

CHECK BAD PTR, LEN
DO UNTIL NO GARBAGE EQUAL LENGTH
END DO
SNAP IN SPACE
GARBAGE LENGTH NOT KNOWN
PUT STORAGE ON GARBAGE LIST
END SEGMENT
SUBROUTINE RESTORE
**********************************************************
*   DYNAMIC MEMORY AND COMMON VALUES ARE READ FROM A FILE
*   INTO MEMORY AS PHYSICAL STRUCTURES. THIS FILE IS
*   CREATED BY SUBROUTINE STORE.
**********************************************************
RESTORE COMMON VARIABLES
RESTORE DYNAMIC MEMORY
1 SUBROUTINE RULES
SUBROUTINE IS NOT IMPLEMENTED YET
SUBROUTINE SAVE
**********************************************************
```

```
*   DYANMIC MEMORY AND COMMON VALUES ARE WRITTEN TO A FILE
*   AS PHYSICAL STRUCTURES. THIS FILE MAY BE USED TO
*   RESTART THE SIMULATION AT THE POINT WHERE IT LEFT OFF.
**********************************************************
SAVE COMMON VARIABLES
SAVE DYNAMIC MEMORY
SUBROUTINE SCRBK(FLDNAM,POS,BOTTOM,LINE)

PURPOSE:   SCROLL A SCROLLED AREA BACKWARD

PARAMETERS:
   FLDNAM  ==> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   POS     ==> POINTER TO NODE THAT IS DISPLAYED ON THE
               CURRENT LINE OF THE SCROLLED AREA.
   BOTTOM  ==> POINTER TO NODE THAT IS DISPLAYED ON THE
               LAST LINE OF THE SCROLLED AREA.
   LINE    ==> LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA.

IF NO LAST LINE OF SCROLLED AREA THEN NO NODES ARE DISPLAYED
IF CURRENT LINE IS BLANK THEN CURRENT LINE IS BELOW LAST
 DISPLAYED LINE.  THEREFORE, SET CURRENT LINE TO LAST
 DISPLAYED LINE.
IF CURRENT NODE IS TOP OF QUEUE THEN SEND MESSAGE ELSE
 SET CURRENT NODE TO PREVIOUS NODE

*** TOP OF SCROLLED AREA ***

WRITE SCROLLED LINE TO SCREEN

*** NOT TOP OF SCROLLED AREA ***

MOVE CURRENT LINE OF SCROLLED AREA UP ONE LINE
SUBROUTINE SCRFWD(FLDNAM,POS,BOTTOM,LINE)

PURPOSE:   SCROLL A SCROLLED AREA FORWARD

PARAMETERS:
   FLDNAM  ==> NAME OF FIELD IN SCROLLED AREA TO SCROLL
   POS     ==> POINTER TO NODE THAT IS DISPLAYED ON THE
               CURRENT LINE OF THE SCROLLED AREA.
   BOTTOM  ==> POINTER TO NODE THAT IS DISPLAYED ON THE
               LAST LINE OF THE SCROLLED AREA.
   LINE    ==> LINE NUMBER OF THE CURRENT LINE OF THE
               SCROLLED AREA

IF NO LAST LINE OF SCROLLED AREA THEN NO NODES ARE DISPLAYED
IF CURRENT LINE IS BLANK THEN DO NOT ALLOW TO SCROLL FORWARD
 I.E. ONLY ONE BLANK LINE AT BOTTOM OF SCROLLED AREA CAN BE
 USED TO INPUT NEW ENTRIES.

*** BOTTOM OF SCROLLED AREA ***

WRITE SCROLLED LINE TO SCREEN

*** NOT BOTTOM OF SCROLLED AREA

MOVE CURRENT LINE OF SCROLLED AREA DOWN ONE LINE
1 SUBROUTINE SCRLINE(POS,FVALUE)
```

PURPOSE: CREATE STRING TO OUTPUT TO THE CURRENT LINE OF A SCROLLED AREA

PARAMETERS:

POS ==> POINTER TO NODE TO CONVERT TO GET FIELD VALUES FOR SCROLLED LINE.

FVALUE ==> FIELD VALUES FOR SCROLLED LINE.

NULL ENTRY, THEREFORE INITIALIZE TO BLANKS

INITIALIZE PORTION OF SCROLLED LINE THAT CONTAINS MAIN NODE AND IT'S ID
INITIALIZE PORTION OF SCROLLED LINE THAT CONTAINS 1ST ALTERNATE AND IT'S ID
INITIALIZE PORTION OF SCROLLED LINE THAT CONTAINS 2ND ALTERNATE AND IT'S ID
SUBROUTINE SETFLAG

PURPOSE: CREATE FLAG NODE. INITIALIZE ALL VALUES TO ZERO.

GET VIRTUAL MEMORY SPACE
INITIALIZE 20 PRINT FLAGS
INITIALIZE THE 3 OPTIONAL PRINT MODIFIERS

INITIALIZE DEBUG PRINT FLAG, DEBUG TIME ON, AND DEBUG TIME OFF
1 SUBROUTINE SIMCTRL

PURPOSE: ALLOW USER TO CHANGE VALUES OF PRINT FLAGS.

FMS TERMINATOR KEYS

CLEAR SCREEN AND PUT SIMULATION CONTROL FORM ON SCREEN

PUT VALUES IN APPROPRIATE FIELDS

GET USER ENTRY

ENTRY WAS A PRINT CONTROL FLAG

IF ENTRY NOT EITHER '0' OR '1' THEN ILLEGAL; RESTORE OLD VALUE; GET ENTRY AGAIN
LEGAL ENTRY - SAVE NEW VALUE; BRANCH TO PROCESS FIELD TERMINATOR

ENTRY WAS AN OPTIONAL OUTPUT TIME - SAVE NEW TIME; BRANCH TO PROCESS FIELD TERMINATOR

ENTRY WAS DEBUG OUTPUT FLAG

IF ENTRY NOT EITHER '0' OR '1' THEN ILLEGAL; RESTORE OLD VALUE; GET ENTRY AGAIN

LEGAL ENTRY - SAVE NEW VALUE; BRANCH TO PROCESS FIELD TERMINATOR

ENTRY WAS DEBUG ON/OFF TIME - SAVE TIME; BRANCH TO PROCESS FIELD TERMINATOR

PROCESS FIELD TERMINATOR

SUBROUTINE SNAPQ(PTR, NWORD, PINS, PINE)

PURPOSE:
INSERT AN ENTIRE QUEUE OF RECORDS INTO ANOTHER QUEUE OF RECORDS. ASSUMES THAT THERE IS A CORRESPONDING BACK POINTER FOR THE FORWARD POINTER. ASSUMES BACK POINTER IS OFFSET FROM ITS FORWARD POINTER BY 1.

```
PARAMETERS:
   PTR     ==> ROOT OF BASE QUEUE
   NWORD   ==> OFFSET FROM PTR TO SORT ON
   PINS    ==> POINTER TO TOP OF QUEUE BEING ADDED
   PINE    ==> POINTER TO BOTTOM OF QUEUE BEING ADDED
```

EMPTY QUEUE - MAKE FIRST RECORD

INSERT BEFORE RECORD

INSERT BEFORE 1ST RECORD

INSERT AFTER LAST RECORD

1 SUBROUTINE UNSNAP(NROOT, NPTR, ISPACE)

REMOVES AN ENTRY FROM QUEUE

SET FORWARD AND BACK POINTERS
IF NODE BEING REMOVED IS NOT 1ST NODE IN QUEUE THEN SET FORWARD POINTER OF PREVIOUS NODE TO NEXT NODE.
IF NODE BEING REMOVED IS 1ST NODE IN QUEUE THEN SET ROOT TO NEXT NODE
IF NODE BEING REMOVED IS NOT LAST NODE IN QUEUE THEN SET

BACK POINTER OF NEXT NODE TO PREVIOUS NODE.

********** UAR ROUTINES **********

INTEGER FUNCTION VALID1

FIELD COMPLETION UAR CHECKS IF VALUE IS BETWEEN 1 AND THE MAXIMUM FOR THE APPROPRIATE FIELD.

GET FIELD NAME OF CURRENT FIELD
GET VALUE AT CURRENT FIELD
CONVERT VALUE TO INTEGER
GET MAXIMUM LEGAL VALUE FOR CURRENT FIELD
CONVERT MAXIMUM LEGAL VALUE TO INTEGER
IF CURRENT VALUE IS BETWEEN 1 AND MAXIMUM LEGAL VALUE THEN RETURN SUCCESS CODE ELSE SEND ERROR MESSAGE AND RETURN FAILURE CODE.

B.2. Program C^3EVAL

There are three distinct elements simulated by C^3EVAL. The first is the C^3 environment. It consists of a set of nodes (command posts), paths (lines of communication), and processing of messages and combat data. Figure 5 gives an example of a command network that could be represented in the model. In this example, Div units would be designated as the combat level units. Each one has specific ground-based weapon systems (tanks, AAA, infantry, etc.) specified for its own forces (Blue) and for its opposing force (Red). In addition, a notional airbase has aircraft (Blue only) which are requested by air tasking order (ATO) messages for specific combat uits at a specified game time interval. These weapon systems and aircraft sorties are the second element.

The third element of C^3EVAL is the combat process. This is modeled by using IDA's anti-potential potential (APP) method of calculating combat value and force ratios. Attrition is calculated by multiplying the APP effectiveness matrix times each unit's weapon system vector. The sequence of events in the model is:

- Limit input messages
- Process messages received
- Create messages based on command post actions
- Limit output messages
- Allocate messages to network
- Put messages into communication status
- Process requests for CAS
- Update combat unit's weapon system
- Calculate combat drawdown

The C^3 portion of C^3EVAL has been implemented in a user-controlled dynamic memory (DM) environment. This DM is a large

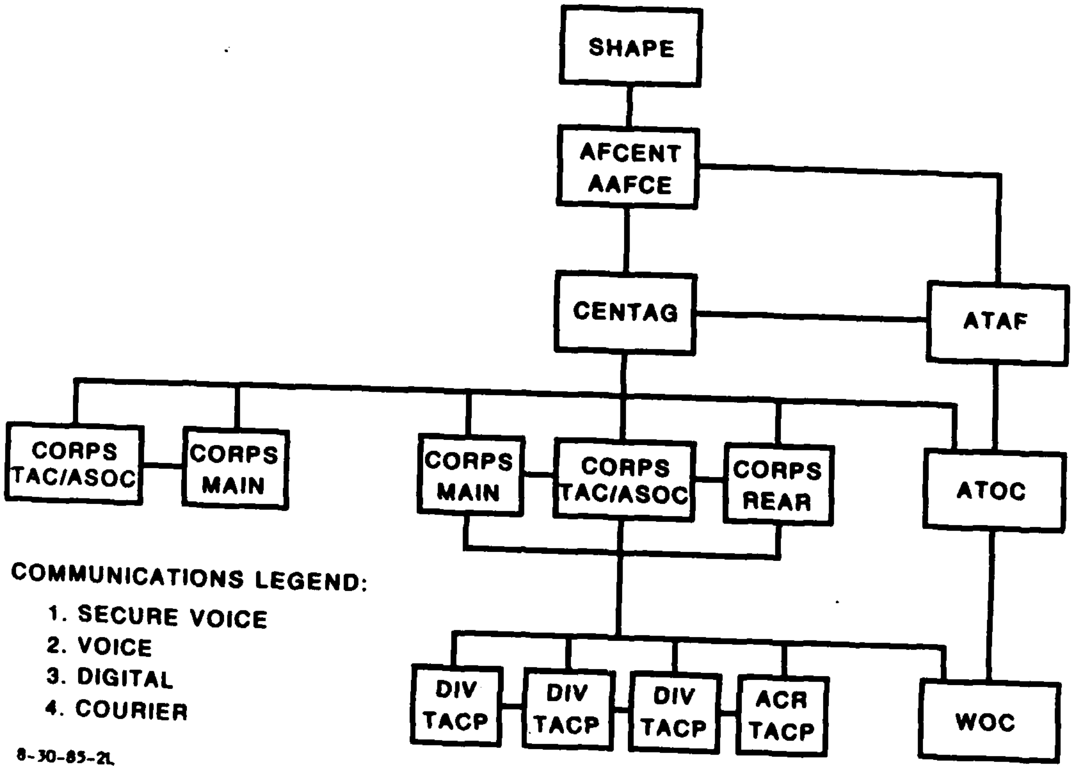


Figure 5: C-3 NETWORK

single-dimension common area that is segmented, allocated, and reused by two simple C^3EVAL utilities. The effect of this implementation decision is a code that provides for a vast variation in the number of nodes, paths, or messages that may be used from different data bases and scenarios without major parameter changes in the model. (Only the maximum size of DM is modified to provide efficient computer memory usage.)

This "dimensionless" code is further enhanced by the use of linked lists of data structures in addition to the standard common arrays. The utilities section of this manual describes the DM subroutines and those that assist in the use of linked lists. The combat force ratio calculations were extracted from the IDA seven methods of evaluating forces based on antipotential potential. This code uses explicit dimensions for the number of different combat systems that will be evaluated. This size was set to 11 distinct combat systems for the current model.

The three functional modules controlled in C^3EVAL as shown in Figure 6 are entered through SUBROUTINES INPUT, EVENTS and OUTPUT. The C^3EVAL process starts with determination of the mode. (Initial run starting at time zero or restart run at Time T). Interpretation of input and output unit numbers and setting of DM to zero is accomplished for time zero runs. DM is zeroed by the utility SUBROUTINE DMINIT. For Time T starts, SUBROUTINE RESTOR is called to reset all dynamic memory and model parameters.

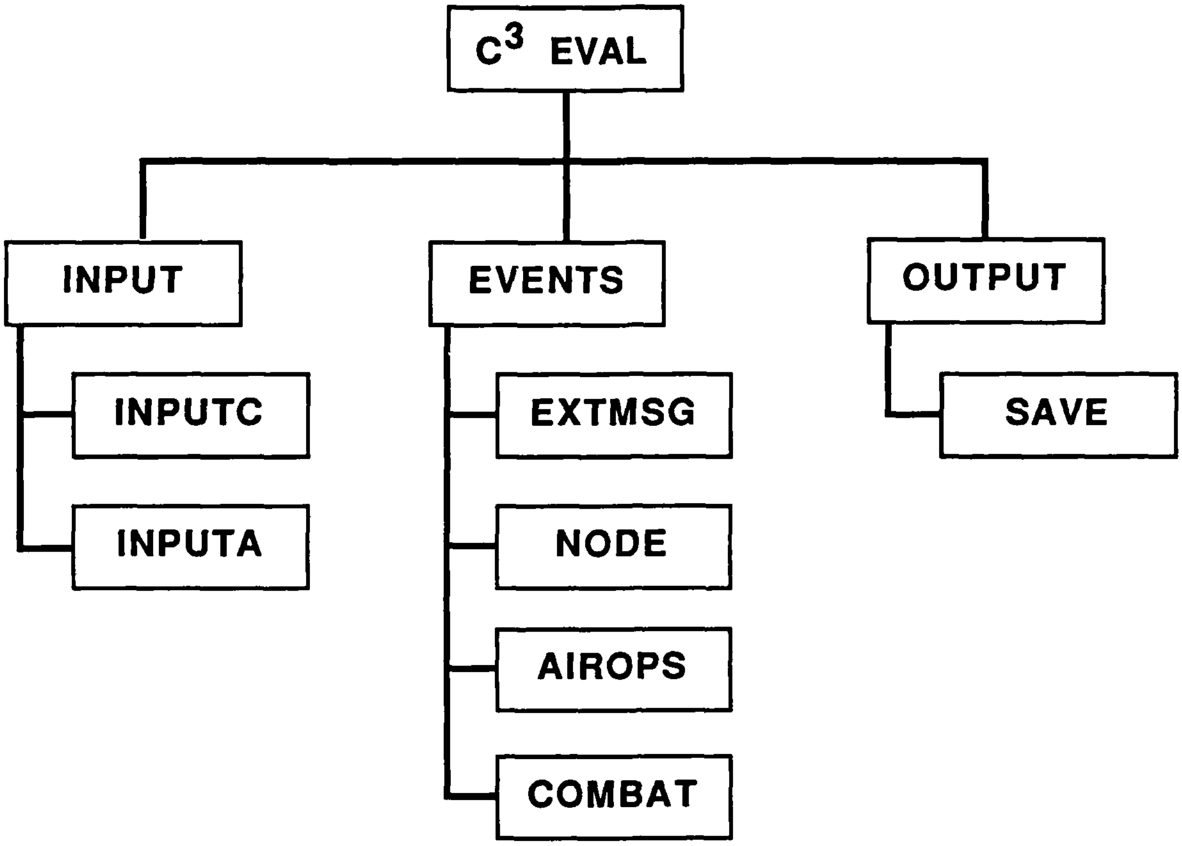


Figure 6. C ^{3}EVAL Modules

10-17-85-15M
Unclassified

SUBROUTINES INPUT and RULEIN are called for time zero runs to establish the decision rules and initial conditions for C^3 and combat. Their operation is described in the next section. SUBROUTINE EVENTS controls all C^3 events, combat calculations, and the game clock. After all requested simulated combat is calculated, summary output is produced by OUTPUT and C^3EVAL stops the run. The hierarchy of subroutines is shown in Figure 7. This figure does not show the attrition module subroutines or the utilities.

B.2.a. Control Module

The main program C^3EVAL performs all the top-level functions of file interface, determination and control of the mode of operation, data input, event simulation and output. The input files are:

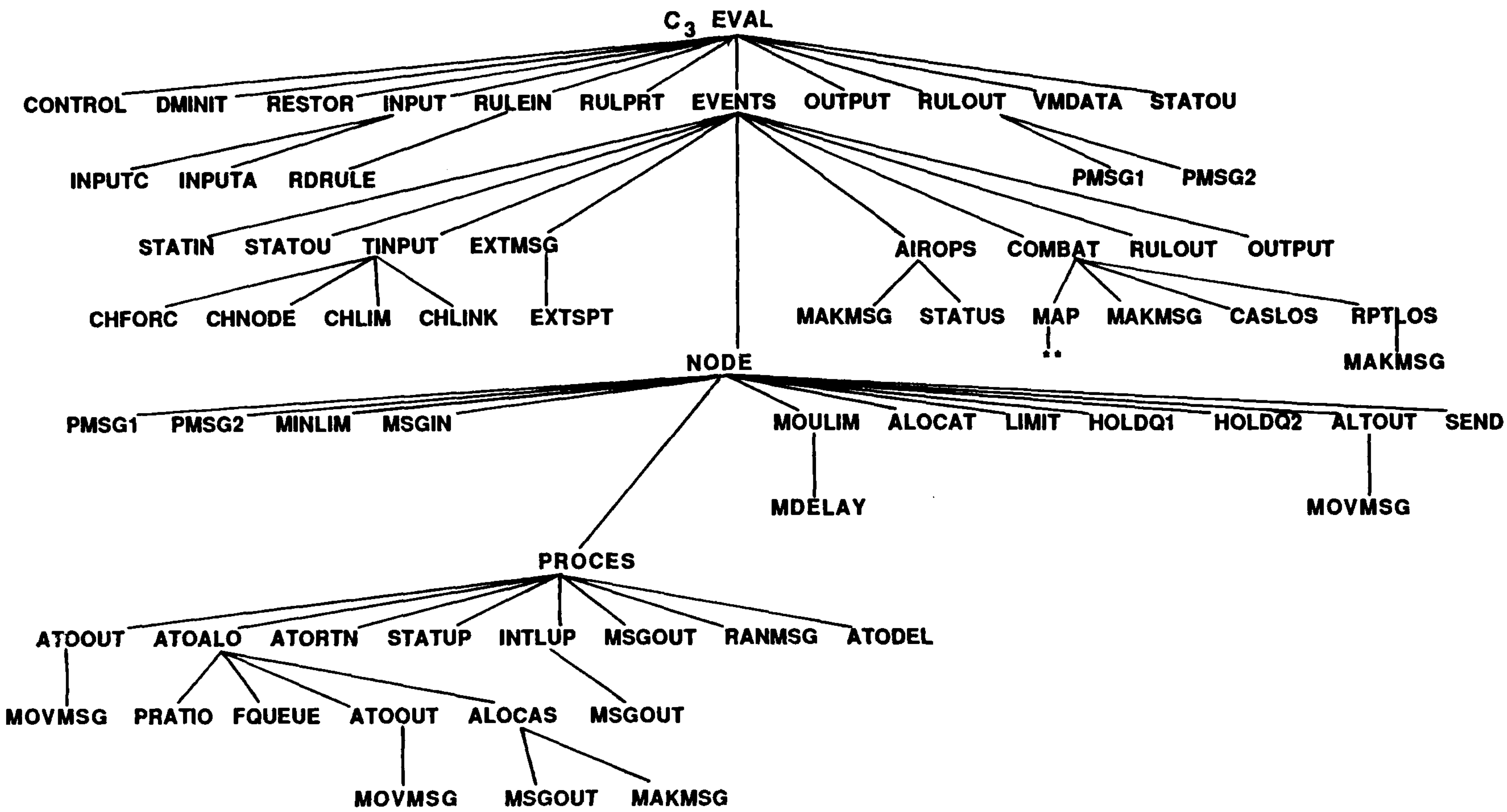
C^3DATA	C^3 descriptions
CBDATA	Ground combat systems data
AODATA	Aircraft operations data
C^3RULE	Decision Rule Data
RESTOR	All data necessary to continue a previous run

The output files are:

C^3ECHO	Echo of input data
C^3TIME	All reports, error conditions, and debug data
C^3SUM	End of run summaries
SAVE	All data necessary to restart from end of a run
TIMET	Message flow data for each time increment
LOSST	Combat losses for each unit for each time increment

Temporary work files are:

C^3NODE	Changes to NODE data to occur during game
C^3LMNO	Changes to node input and output limits to occur during the game.
C^3LINK	Changes to link capacities to occur during the game.



No'ə
Utilities not shown
**Attrition not shown

Figure 7. C ^{3}Eval Subroutine Hiarchy

10-17-85-9 M
Unclassified

B.2.a(1). Subroutine CONTRL

CONTRL is called by C^3EVAL to read the print and debug parameters and to read the preamble from the C^3DATA file. The preamble is used to document the contents of the file, to easily identify variations of a basic scenario and to assist in relating other files and output to a specific run. The first 40 characters of the top line of the preamble will appear as the subtitle on all graphics. The preamble may contain as many lines of 80 characters as necessary. It must be completed with "END " as the first four characters in the last line. Contrl copies the preamble to files C^3TIME, C^3SUM and TIMET. The definitions of the flags are shown in Figure 8 which is a PreProc screen. The flags are echoed to file C^3ECHO.

PRINT CONTROL FLAG	
ALL MESSAGES AT ALTERNATE NODE	0
ALL MESSAGES ON INPUT QUEUES	0
ALL MESSAGES ON OUTPUT QUEUES	0
ALL MESSAGES ON FUTURE QUEUES	0
ALL MESSAGES BEING HELD	0
ALL MESSAGES DELETED	0
STATUS OF RULE STRUCTURE	0
CAS TAKE OFF SCHEDULED	0
not assigned	0
not assigned	0
TRACKED MESSAGES AT ALTERNATE NODE	0
TRACKED MESSAGES ON INPUT QUEUES	0
TRACKED MESSAGES ON OUTPUT QUEUES	0
TIME T OUTPUT ON FILE 14 REQUIRED	0
COMBAT LOSS VECTORS	0
FORCE RATIO CALCULATIONS	0
RULE STATUS AT FINAL TIME	0
not assigned	0
RANDOM PROCESSING REQUIRED	0
USED INTERNALLY FOR SUM OF FLAGS	0

OPTIONAL OUTPUT	
SPECIFIC NODE	0
START TIME	0
STOP TIME	0
DEBUG OUTPUT FLAG	0
START TIME	0
STOP TIME	0

(NOTE: FOR FLAGS 0 --> OFF
1 --> ON)

HIT <RETURN> TO RETURN TO MAIN MENU

Figure 8: PRINT AND DEBUG PARAMETERS

B.2.a(2). Subroutine DMINIT

DMINIT is called by C^{3}EVAL when a game is to be started at time zero. It prepared DM for use by initializing the next available memory locator (MPTR) to 1 and setting all DM and the garbage pointer to zero. The garbage pointer is the root of the reuseable memory block queue.

B.2.b. Input Module

Input data is of two types. The basic scenario data (or time zero data) is read by the Input Sub-routines and its subordinate routines. See Figure 9. The event (or Time T data) is read by subroutines under control of the Events. See Figure 10. The Block Data FRatio is included here because of its function. It is not called by Input, of course.

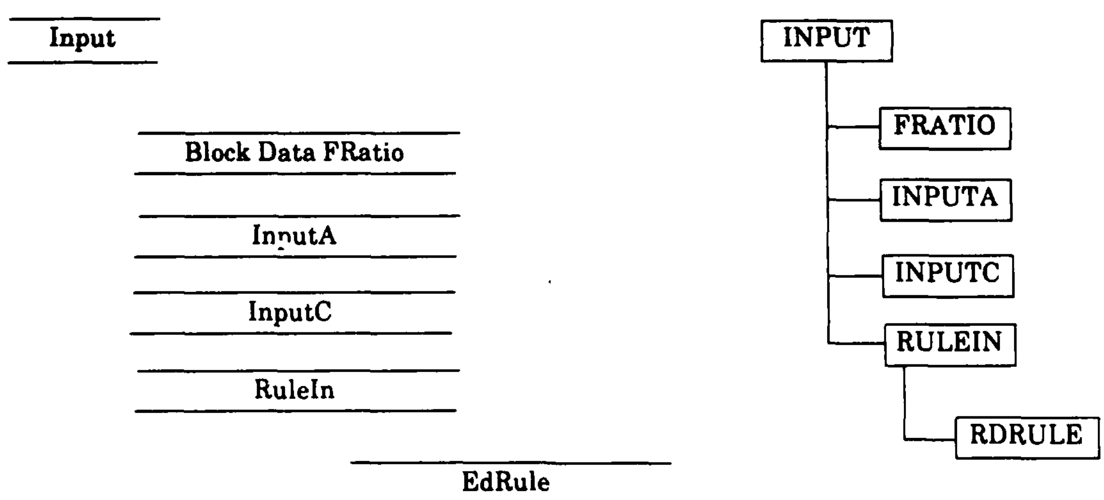


Figure 9: SCENARIO INPUT SUBROUTINES

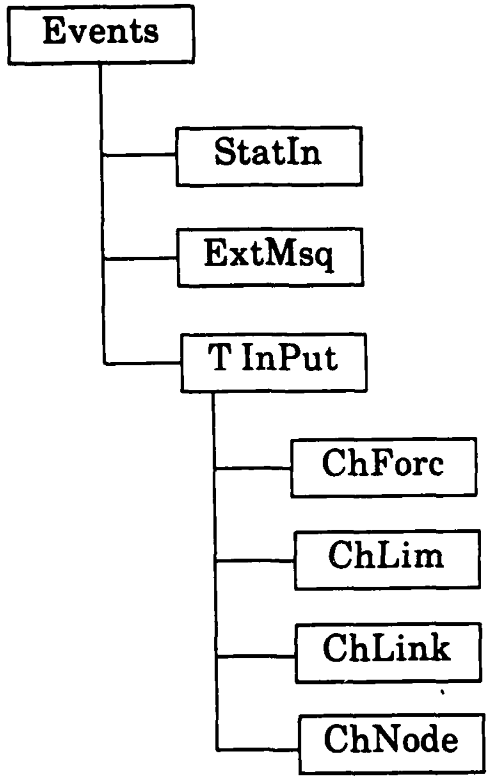


Figure 10: TIME T INPUT SUBROUTINES

B.2.b(1). Subroutine Input

SUBROUTINE INPUT reads node, link, and limits data in a manner that allows the user to build this file in alternative styles, depending on the form most applicable to the analysis process. Data is read from a file identified by the variable INP which is set to one in C^3EVAL. The general form is to read a set name line, a header of 80 characters, and then a line of data. The first four characters are reserved to identify the type of input involved. All data lines must have the first four characters blank. This implies that this data line belongs to the set identified by the last set name read. Set name lines must have one of the following set names in the first four characters: NODE, LMNO, LINK, or PROC in order to input that type of data. Anything other than blank as a set name will cause the input of the data set to end (normally set to LAST). This allows the

analyst to put all node data togeter in file INP or to have a mixed sequence of set types and normally grouping NODE, LMNO, LINK, and PROC data for each unit identified. A header line must always follow immediately after a set name line. It may be used as comments about the data or left blank.

In the program, the NODE set is located at the top of the subroutine. The first time a new node number is read a NODE data structure is created and unit number, identifier and type are set. The same node number will be required on each data line that identifies a different destination with which the node can communicate. Each time a new node number is found it is linked into the node queue which has its root in NODE1 in COMMON/C^3/. A destination (DEST) data structure is created for each destination identified. The DEST data structures are linked together in the node's destination queue. The DEST data structure is initialized with its unit identifier and one or two destinations is the node's commander, the commander's element in the node data structure is set to that destination's unit identifier. If a destination is a subordinate of the node, the subordinate flag is set in the node's DEST block. If the input value for the subordinate flag is equal to 2 then a subordinate status data structure is created and snapped into the node's subordinate status queue.

The NODE data is read by format (A4, 8I5, 3A4, 24A1). The first field is the data set type (NODE on the first line and blank on all other). The 8 numbers are: time, unit, unit type, destination, alternate node 1, alternate node 2, commander flag and subordinate flag. The flags indicate the relationship of the destination to the unit. The 3A4 field is the unit identifier and the remaining data are comments (normally, the destination's identifier). If time is greater than 1, the data line is written to file C^3NODE to be processed at the future time.

The next set in Input is LMNO, the LiMits for iNput and Output. It is read by format (A4, 6I5, 45A1). The first field is the data set type. The six numbers are: time, unit and four message limits. If time is greater than 1, the data line is written to file C^3LMNO to be processed at the future time. The limits are: hold for input, delete for input, hold for output and delete for output. The comment field may be used to document the line (i.e., curtailed operation due to direct attack).

The next set in Input is LINK, which indicates the capacity of each type of communication between two nodes. The format used is (A4, 4I5, I10, 46A1). The first field is the data set type. The five numbers are: time, unit, unit communications type, and capacity. The last field is for comments. If time is greater than 1, the line is written to file C^3LINK. Notice that a link has a node at each end and that these nodes must have been identified before any link information can be given. A particular link should be identified only once, and the input procedure for the LINK set is indifferent which node of the link is given first or second. A link line is required for each type of communications desired between each node pair. The LINK process finds the data structure for the node identified on the link input line. Then a LINK data structure is created and linked into the destination structure for the node. The type of link and its capacity are set in the link structure. This process is then repeated for the node on the other end of this link. The final data set recognized by Input is PROC. This type is not currently used and exists for compatability to earlier versions.

When a non-set name is found by Input, it branches to input combat data. This is done by a call to INPUTC. Then air operations data is input via a call to INPUTA.

At this point, all data has been read in, but there is still some initialization required for all node and destination structures. The first step is to replace the commanding unit identifier with a pointer to the commander's node data structure. Next, the unit type in all destination data structures are initialized by finding that data in the destination's node structure. Finally, the alternate/s are located by their unit number and the destination data elements for alternates are set to pointers to the alternate's node structure.

B.2.b(2). Subroutine Inputa

This routine reads in air operation's data from file AODATA. This file has a documentation preamble which is followed by a comment line and one or more lines containing aircraft allocation parameters. This line is read with format (A3, 6X, 6I6, 10A4). If the first field is blank, the data is processed; if it is equal to "LAS", the data set is terminated. The six numeric parameters are; allocating node, message number that the parameters are to apply, initial allocation time, number of periods to be used in the allocation process, the maximum number of sorties for a flight, and the maximum sorties to be allocated throughout the specified periods. The last field is a comment field. Inputa finds the node structure indicated, creates and initializes an allocation structure, and snaps it into the nodes allocation queue.

The next data set in AODATA identifies the CRC node. A documentation header is read. The first data line contains the CRC node number, the alert time, enroute time, the minimum number of aircraft that constitutes a CAS sortie, and the probability of survival enroute of the aircraft on a CAS mission. The following data lines must be blank in the first three columns and contain the WOC node identifier, the aircraft type, the time the aircraft will be available, and the number of aircraft. THe AIROPS input

is completed when the first three columns are set to "END".

The program creates the CRC structure and sets NCRC in COMMON/C^3/ to this data structure. It stores the CRC data values in that structure. For each unique WOC read, a WOC structure is created and the aircraft availability data is stored in the READY and RDYQ structures.

B.2.b(3). Subroutine Inputc

This routine reads in the combat weapon system values from CBDATA for each combat level node. This file has a documentation preamble which is followed by the generic red unit data set. Each data line contains a red unit type, eleven values for the weapons systems and the posture for this type unit. This data set is completed when unit type is equal to "99999". A red table of equipment data structure is created for each line of data and snapped into the generic red unit queue with root in variable NREDTE in COMMON/C^3/.

The last data set read by Inputc is the combat weapon systems data. The first line read is a comment line to assist the analysts to identify the scenario data it represents. Each data line contains a mode identifier, unit posture, and the number of weapons in each combat system type. It then finds the NODE data structure that is specified, creates a CMBT data structure for that node, and stores the number of systems in that structure. When the NODE identifier is found to be "99999," the input is completed.

B.2.b(4). Subroutine RdRule

RdRule reads in the command post (node) rules which consist of three data sets. The data is read from file C3RULE. The file has a data preamble followed by three lines of heading for the rule parameter data set. This set is read by format (I5, I10,

4I5, 2X, 3A4). The variables read are; Rule number, type unit that uses the rule, time required to perform the processes, the minimum messages required to initiate the process, an indicator of a periodic or reaction-type process, and the start time for periodic processes. If the minimum messages required is zero, then the process is done for each current message. The last field read on each line is the title of the decision rule. This data set is completed when a rule number is zero.

The next data set read is the input message data. It has two lines of heading and is read by format (I5, I10, 3I5, 2X, 3A4). The variables read are: Rule number, unit type that originated the message, message type, maximum age for the message to be useful and a flag to force a message to be used only once. This is to keep an input message that is retained for more than one period from generating the same output message more than once. The last field is the title of the message. The data set is completed when a rule number is zero.

The final data set is the output messages to be created by a rule. It has two lines of heading and is read by format (I5, I10, 2I5, 3(I5, I6), 2I2, 2I5, I2, 2X, 3A4). The variables read are; rule number, destination unit type, output message number, priority, three sets of link type and capacity required, flag if destination is commander only, output flag, two alternate destination unit types, the maximum time the message will remain active in the communications network, and the title of the output message. NOTE: This data is read into 8 arrays in COMMON/RULE/. These arrays currently have a maximum of 300 entries. The output flag for random process types is also used as the minimum time between report generations.

B.2.b(5). Subroutine RuleIn

The subroutine creates the data structures OMP, IMT, and MSG for each node. The parameters for the process, input messages and output messages are read from file C^3RULE bySubroutine RdRule. RULEIN processes each rule in array IRULE until IRULE (1,N) is equal to zero. For each rule number, a queue of generic messages is created from the MSGOUT data from C^3RULE. This queue has its root by rule number in the MSGQ array. After this queue has been created, an output message process structure is created for each node of the type indicated as an originator of the process. This OMP structure will have a pointer to the queue of generic messages that will be created each time the process is successfully initiated during the game. This success is based on receiving the desired information at a node. This information is indicated in an input message table (IMT). The IMT is created by RULEIN from parameters from C^3RULE and is attached to the OMP data structure. At the completion of initializing all rule processing, RULEIN calls SUBROUTINE RULLOUT to echo the data structures to output.

B.2.b(6). Subroutine StatIn

StatIn is a part of the input module because it initializes the data values in a commander's status data structure. This data is the perceptions by the commander of his subordinate unit's strengths and combat postures. The initial perceptions are based on the input numbers of weapon systems for the subordinates. The perceptions of the subordinates for (red) are set to the first generic red unit that was read in and the foe unit title is set to "uninitialize".

StatIn is called by the events subroutine prior to entering the combat time loop. It initializes only the nodes that have status blocks created by Input.

B.2.b(7). Time T Input Sub-Module

C^3EVAL has the ability to accept scenario based messages created by the user and to modify combat units' force structures, command posts' input and output processing limits, communications paths capacities and preplanned changes in the command structure. These changes are indicated in input data by specifying the desired game time for them to take effect. Subroutines ExtMsg and TInPut are called by subroutine Events during each game cycle. The other subroutines in this section; ExtSpt, ChForc, ChLim, ChLink and ChNode are used to support these routines.

B.2.b(7)(a). Subroutine ExtMsg

This subroutine reads in all external messages from C3DATA. These messages must be in time-sorted order. The time that a message is to be used is read into MIN(1). If this is some future time, processing is returned to EVENTS. When messages are to be processed, a MSG block is obtained from DM and filled with the input elements. If an ATO is indicated in the message, a DATA block is obtained, the ATO data read and transferred to the block. The ATO is attached to the message via the pointer PDATA. Next the node that is to receive the message is found and the message is put on the node's input message queue. The format for messages is (9I6). The variables read are; message times; message type; unit type of originator of the message, destination unit, unused field, output flag, additional data flag, priority, and time message was created. The format for additional data is determined by the message type. If the message type equals 3136, the data is read by subroutine ExtSpt.. Otherwise, it is read by format (6X, 6I6) and the variables are: support unit, earliest support time, latest support time, type of aircraft, and number of aircraft.

B.1.b(7)(b). Subroutine ExtSpt

ExtSpt reads additional data for messsage type 3136 from C^3DATA. The format used is (6X, 4I6, 5X, 3A4, I1). The variables read are: unit, time, type of foe, posture of foe, foe's title, and flag to indicate additional data on the next line. ExtSpt creates an Intel Report Data Block for each data line and snaps the queue into the additional data pointer in the message.

B.2.b(7)(c). Subroutine TInput

Subroutine TINPUT inputs changes to characteristics during a specified time T. The characteristics that can be changed are: number of weapons and posture of a unit, input and output message limits at a particular node, a node's commander or subordinate, and the message capacity of a particular link type between two specified nodes. The reinforcement changes are read from a formatted file. The node, limit, and link changes are read from three unformatted files. Future changes are input and held until the specified future clock time. *There may be more than one* change per characteristic at any given clock time. The input stream must be in time ordered sequence. When an update time occurs, the appropriate subroutine is called to process the change.

B.2.b(7)(d). Subroutine CHFORC

Subroutine CHFORC changes the combat values for a specified unit number for a blue combat unit and its corresponding red combat unit. If the unit number of the blue combat unit does not exist (not found in NODE queue) then an error message is produced. Otherwise, the posture of the red and the blue combat units are updated. The number of red and blue weapons are updated for reinforcements. Note: A blank line must be given for a side that has no change.

B.2.b(7)(e). Subroutine CHLIM

Subroutine CHLIM changes the input or output message limit at a specific node. The message limits are: maximum number of input messages that can be received at a particular node during one time increment, maximum number of input messages that can be received or held at a particular node, maximum number of output messages that can be sent from a particular node during one time increment, maximum number of output messages that can be sent or held at a particular node. Note: Both must be given even if one does not change.

B.2.b(7)(f). Subroutine CHLINK

Subroutine CHLINK changes the maximum capacity of that particular link type between two specified nodes. The values passed to CHLINK are: clock time change is to be made, unit identifier of node at one end of link, unit identifier of node at other end of link, link type, and new maximum path capacity.

B.2.b(7)(g). Subroutine CHNODE

Not implemented yet.

B.2.b(8). Block Data FRatio

This routine contains the data used by the force ratio calculation sub-module. This data includes the names of the weapons system types and the engagement rates. There may be three sets of engagement rates for Red and Blue. Variable I1 in COMMON/BLUE identifies the index weapon systems and the number of weapon systems is set. Then the allocation matrix for a generic unit is set for Red and Blue. Finally, the Red and Blue probability of kill matrices is set.

B.2.c. Events Module

The EVENTS Module contains all of the C^3 combat, air operations, and Time T Input/Output. This section will discuss the functional subroutines in this module. The utility subroutines that are used in this module will be described in section C. The structure of the event processes is shown in Figure 11.

B.2.c(1). Subroutine EVENTS

The Events Subroutine is an executive routine that controls the game clock and sequence of events. For each time period, external events are obtained by TInPut and external messages are obtained by a call to ExtMsg (see Section B). Messages are received at command posts, processed, and new messages generated by the call to NODE. Close air support sorties are allocated, scheduled, controlled while airborne, landed and rescheduled for COMBAT. Game time is incremented until the end of game time is reached when processing is returned to C^3EVAL. Subroutine Events checks the user's indicator for time T output data. If this is requested, Subroutine Output is called at each time interval.

B.2.c(2). C^3 Sub-Module

This sub-module represents the command post actions (as specified by the decision rules) and the communications between command posts. Communications are represented by data specified paths between nodes and the capacity of each path to carry message traffic based on priorities. Messages can be generated by user (external) input, by random occurrance based on decision rule parameters or in response to internal events such as receipt of a message or the change in a force ratio beyond input limits. Messsages may be sent, delayed or deleted from the network. Each path capacity may be modified at any time interval to represent

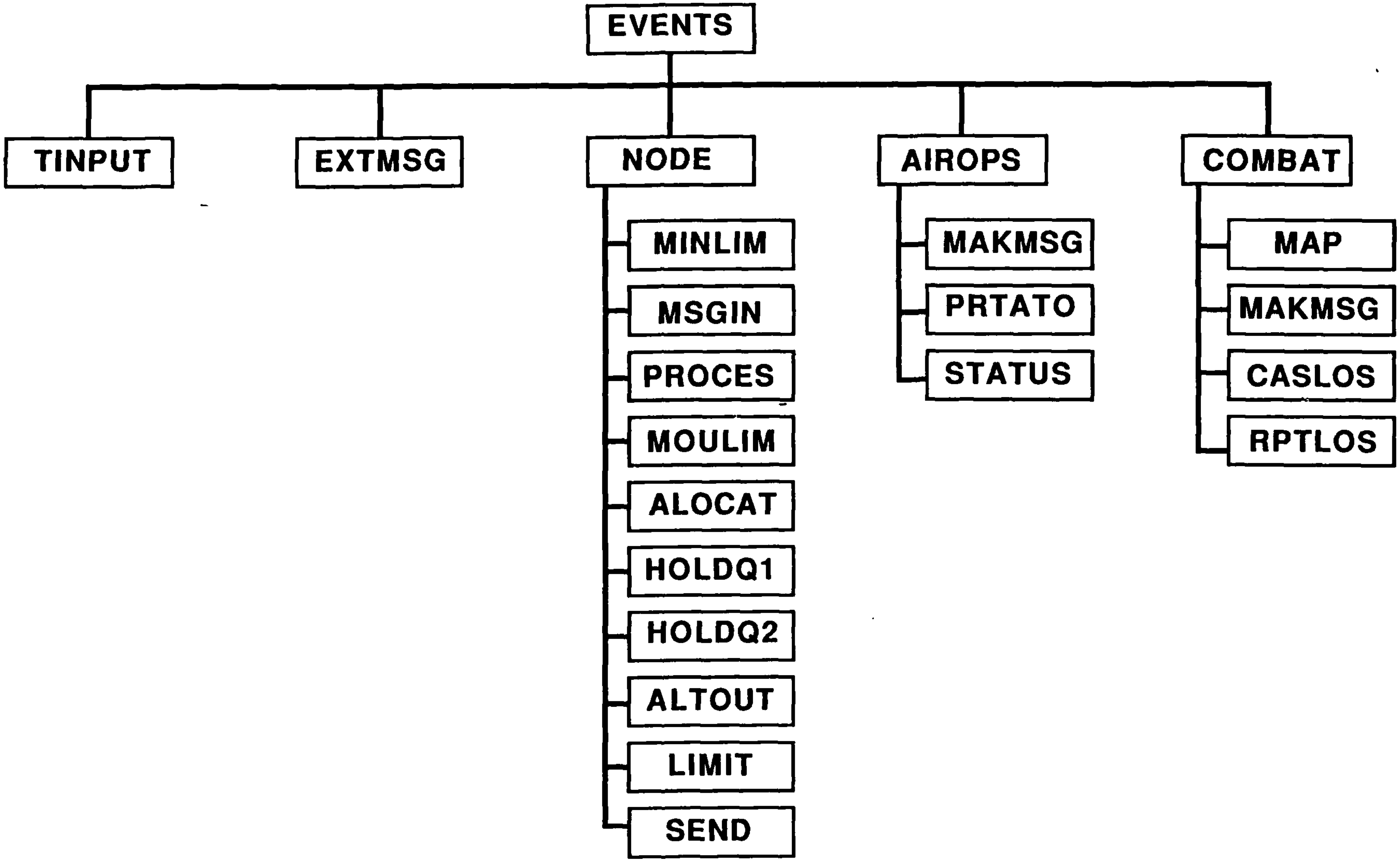


Figure 11. Structure of the Event Processes

10-17-85-13M
Unclassified

direct attacks, E W, etc. Each node has input and output limits on the number of messages that can be processed during each time cycle. Designated nodes will maintain perceptions of their subordinate's capabilities and the opposing forces via messages received. Allocation of support weapons are based on rules applied to these perceptions. The communications network will attempt alternative communications types and routings to send mesages that may be delayed. The details of message creation, movement, arrival and destruction are made available in continuous and summary form. All C^3 processes are accomplished by Subroutine Node and the routines that support it.

B.2.c(2)(a). Subroutine NODE

This subroutine has three main sections to model a commmand post's C^3 events. The first section processes the input messages that are on its input message queue. In this section, each message is counted and checked to see if it is addressed to the node or has been sent to the node to be routed to its final destination. Rerouted messages are simply moved to the output hold queue for communication handling. If the destination for the message is not on the node's destination queue, the misrouted message is deleted and an indication of this action is put in standard output. All input messages at each node are reviewed by subroutine MInLim where input limits are applied. MInLim is also called after processing to put the messages held due to limit on the node's input queue for the next time cycle. Subroutine MsgIn is called to process the messages that meet the limits.

When all incoming messages have been processed, the decision rules section is entered. This section models all processes that have been specified by input for this node type. A process may be periodic or based on reactions to input messages. If it is periodic and the time for the process is the current time, SUBROUTINE PROCESS is called to generate the output messages. If a

process is reactive, then the input messages by type, originator type, and number are checked to see if sufficient current information is available to complete the process. If the processs is able to be completed, then SUBROUTINE PROCESS is called to generate output messages where rules have been met. After all processes have been completed at a node, Subroutine MouLim is called to limit the number of messages that will be output to the network.

After all processes have been completed for all nodes, the communications section is entered. This process is modeled by a series of subroutine calls that review various aspects of allocating messages to available communications links capacities. The sequence of subroutine calls for message allocation is shown in Figure 12. In that figure, SUBROUTINE LIMIT is shown after each allocation routine because it calculates the effects of the two-way communications limit, while the other routines use the simpler one-way limit. The first acceptable communications link found is used by each routine. Therefore, to model this complex process, each destination and link must be checked separately. In addition, messages may be bumped by higher priority messages but may have sufficient priority to bump other messages on different destination/link combinations. Therefore, each process is tried twice. At the end of this sequence, all messages that are successfully communicated are moved to the receiving node by Subroutine Send and process control is returned to SUBROUTINE Events.

B.2.c(2)(b). Subroutine AloCAS

This routine allocates sorties to requests for CAS. It is called by AtoAlo which establishes the number of sorties which can be allocated in accordance with the existing plan and prioritizes the requested received. AloCas approves or disapproves the requests based on the number of sorties specified by AtoAlo.

```
*PRIMARY DESTINATION
   ALOCAT,LIMIT                          by primary links
   *FIRST PASS FOR ALTERNATE LINKS
      HOLDQ1,LIMIT                       1st alternate link
      HOLDQ2,LIMIT                       2nd alternate link
   *SECOND PASS FOR ALTERNATE LINKS
      HOLDQ1,LIMIT                       1st alternate link
      HOLDQ2,LIMIT                       2nd alternate link
*ALTERNATE DESTINATIONS
   ALTOUT,LIMIT                          1st pass, all links
   ALTOUT,LIMIT                          2nd pass, all links
*FIRST ALTERNATE DESTINATION
   HOLDQ1,LIMIT                          1st alternate link
   HOLDQ2,LIMIT                          2nd alternate link
*SECOND ALTERNATE DESTINATION
   HOLDQ1,LIMIT                          1st alternate link
   HOLDQ2,LIMIT                          2nd alternate link
*RECHECK ALL POSSIBILITIES
   ALTOUT,LIMIT
*SEND MESSAGES
   SEND
```

Figure 12: MESSAGE ALLOCATION SEQUENCE

Approved requests are forwarded as messages to the wing operations center. Disapproval messages are sent to the requestor. The count of sorties approved under the plan is updated.

B.2.c(2). Subroutine Alocat.

This subroutine is used by SUBROUTINE NODE to initate the allocation of message traffic to the elements of the communications network. Each message on the future message queue that is to be sent during the current time slice is moved to the

appropriate destination link's hold queue in priority order. When this is complete, the hold queue is moved to the send queue and the send queue is checked to see if there is more message capacity required than the link can carry during the time slice. Any messages that are over the link's capacity are moved back to the hold queue. (Note: This is a one-way check. This means that this part of the allocation algorithm assumes that this node could use all available communications capacity when in fact, there is another node at the other end of the link with messages to send as well. This condition is adjusted in SUBROUTINE LIMIT, which is called by NODE immediately after ALOCAT is finished.)

B.2.c(2)(d). Subroutine AltOut

Allocation of message traffic in a busy network can be an involved process. The algorithm used by C^3EVAL is intended to model the actions that would take place in a message center. It follows capacity limits of the network's links and message constraints of priority, link types, and acceptable routings. The number of parameters that are involved are indicated in Figure 13. The message may have 9 possibilities (3 destinations x 3 link types). The node may have several destinations each with its own set of link types The primary destination and link is tried by SUBROUTINE ALOCAT. The HOLDQ sub-routines allocate to a specified destination using either alternate link 1 or 2. SUBROUTINE ALTOUT allocates by any acceptable link (including the primary link) to message and node alternate pairs. This is accomplished by four successive calls to the utility SUBROUTINE MOVMSG. ALTOUT sets the nodes two way allocation limit flag off because this limit may be exceeded by its process.

B.2.c(2)(e). Subroutine AtoAlo

AtoAlo is called by Proces for each node that is processing requests for CAS (message types 2900, 3000, 3400) and has a non-

```
NODE                                  MESSAGE

DEST                                    DEST
   PLINK          LINK1                 ALT1
                  LINK2                 ALT2
                    .                   LINKP
                    .                   LINKA1
                    .                   LINKA2
                  LINKn

ALT1
   PLINK          LINK1
                  LINK2
                    .
                    .
                    .
                  LINKn

ALT2
   PLINK          LINK1
                  LINK2
                    .
                    .
                    .
                  LINKn
```

Figure 13: PARAMETERS IN COMMUNICATIONS ALLOCATION

null pointer to its allocation parameters queue. AtoAlo processes all pending requests with one call for each message type. When the first request is received for a unit, AtoAlo creates an entry in an allocation queue. Any additional requests that are active for the same unit are consolidated into this entry. After all appropriate requests have been consolidated, routine PRatio is called to calculate the processing nodes perceived force ratios for all the subordinate units.

Then FQueue is called to create a separate set of pointers in the subordinate's status structure sorted by perceived force ratios. The allocation parameters are searched for the appropriate message type. If it is not found, routine ATOOUT is called to process the air requests without allocation

restrictions. If the allocation parameters are found, the number of sorties available for allocation as a function of the allocation time period are calculated by a straight line technique and passed to routine ALOCAS for allocation to requests. The final step is to update the log of sorties allocated.

B.2.c(2)(f). Subroutine AtoDel

Routine Proces calls AtoDel at the completion of each process to delete all ATO requests and reply blocks that belong to the process.

B.2.c(2)(g). Subroutine AtoOut

This routine is called by Proces and AtoAlo to process requests for CAS without allocation restrictions that are carried by message types 2900, 3000 and 3400. AtoOut checks each entry on the nodes ATO queue to find the ones that match the current process numbers. Routine MsgOut is called for each acceptable ATO and the number of sorties approved for the requesting unit is increased in the node's status block.

B.2.c(2)(h). Subroutine AtoRtn

AtoRtn is called by Proces to pass along in accordance with the decision rules, notification of receipt of requests for CAS. AtoRtn checks each entry on the node's ATO return queue to find the ones that match the current process number. Routing for the message is obtained from the ATO's return destination list. A message structure is created and filled in accordance with the generic output message format for the rule. Alternate node and communications data are set for the unique parameters of the node and the message is put on the nodes' send queue or on its future message queue based on time to create the message.

B.2.c(2)(i). Subroutine FQueue

Routine AToAlo calls FQueue to create a threaded list through a node's status structure. The order of the list is based on the force ratio in each status structure. FQueue first sets all pointers in this list to the "unset" value of 5. Then it traverses the node's status queue to find the maximum force ratio in blocks with an unset value. When the maximum is found, it is snapped into the force ratio queue which removes the "unset" value. This process is repeated until all status structures are sorted.

B.2.c(2)(j). Subroutine HoldQ1

The communications section of SUBROUTINE NODE uses the HOLDQ1 and HOLDQ2 subroutines to move messages from a hold queue to an alternate communications link if the message has a sufficiently high priority. Each message has the capability of having a primary and two alternate communications type links and a primary and two alternate destinations. The alternate links and their capacities are specified in the "generic messages data structures" in COMMON/RULE/. This specification is necessary because the acceptable communications type is a function of message type and the capacity required is a function of link type and message type. Destination type is also a function of message type. However, the explicit destination node identify is a function of the message originating node. Therefore, COMMON/RULE/ contains unit type for destinations and the specific node identities are filled in by SUBROUTINE MSGOUT when a message is automatically created. User input messages are placed directly into the node's input message queue by SUBROUTINE EXTMSG and alternate links and node data is not used for this type of message.

HOLDQ1 has the function of placing each message on the hold queue onto a link that matches the type specified as the first alternate type link. It has the capability of using either the primary or one of the alternate destinations. This selection of destinations is made in NODE and passed to HOLDQ1 by the calling sequence parameter IPASS. IPASS equal to zero means the primary destination should be used. When IPASS is equal to 1 or 2, then the corresponding alternate destination is used. HOLDQ1 also bumps all messages of lower priority that exceed the "one way: link capacity as a result of moving a message to its alternate communications link. Note that "bumped" messages are returned to the hold queue for the message's primary destination and primary link.

The sequence of operations of HOLDQ1 is as follow:

- Set allocation flag for this node to off. This flag indicates that a two-way limit has been completed for this node. NOLDQ1 will modify the results of any previous two-way limiting and therefore this action will have to be repeated. See SUBROUTINE LIMIT for a description of this flag's implications.
- Identify the appropriate destination based on IPASS value.
- Find nodes link with type equal to alternate link one of message (if it exists).
- If message priority is higher than a message on the send queue and capacity exists for this message, then move message to the appropriate (priority order) position in the queue.
- Set alternate communications flag to 1.
- Move any lower priority messages on the send queue to their hold queue if they exceed the "one way" capacity check.

B.2.c(2)(k). Subroutine HOLDQ2

This subroutine operates the same as HOLDQ1, except that it uses the message's second alternative communications link for the desired communications type. See HOLDQ1 for description of operation.

B.2.c(2)(l). Subroutine IntlUp

Routine Proces calls IntlUp when the message type is 3136, intelligence update. IntlUp updates a commander's perception of a subordinate's combat foe. IntlUp checks each entry on the commander's spot intel queue to find the ones that match the current process number. The perception is updated if the spot report has data that is later than the current perceptive data. If the spot report is most current, all previous perceptions of foes are deleted and the current report of one or more foes are entered in the commander's status structure. The titles of the foe units are for specific units. The unit type will be used to refer to generic unit data for foe unit strengths.

B.2.c(2)(m). Subroutine Limit

This subroutine is used after each of the allocation subroutines: ALOCAT, HOLDQ1, HOLDQ2, and ALTOUT. Its function is to insure the two way limit of a communications link is not exceeded in the allocation process. LIMIT does this by starting with the root NODE and checking the link limit for each link for each destination node. In order to be efficient when the root node and each subsequent node has been processed completely, a flag is set to 1 in each destination data structure. When the key node (the node which will have all of its destinations processed next) starts to process a destination, the allocation flag indicates that the two way check has already been accomplished with the destination. For example, if the root node is

number 1 and the next nodes are 2 and 3 in the node queue, the process starts with 1 as the key node. LIMIT checks each link between 1 and 2 and sets the allocation flag. Then it processes 1 and 3 and sets that flag. The next step is to make 2 the key node. It is not necessary to process 2 to 1 because this was done previously. Therefore, in this example the links between 2 and 3 would be checked and the process would be finished.

The capacity check is accomplished in message priority order by comparing the next message at the key node to the next message at the other end of the link. When the capacity is exceeded by a message, all of the remaining messages on both nodes' send queues are moved (in priority order) to the messages' "home" hold queue. The "home" hold queue is the hold queue for the actual destination (not alternate) and on the primary link. The final step is to set the allocation flags in both destinations data structure.

B.2.c(2)(n). Subroutine MakMsg

This subroutine creates messages relating to ATOs. This type message differs from most message types in that it has an additional data structure (ATO) attached to the standard message structure via the pointer PDATA. The message priority is set at 1 and the maximum time for each message to be on the network is 3 time increments.

B.2.c(2)(o). Subroutine MDELAY

Subroutine MDELAY is used by subroutine MOULIM to determine which messages will be sent, held or deleted. Alternate and future messages are ignored. MDELAY receives from MOULIM the value of the maximum priority level to be sent (JP1) and the number of messages to be sent from that priority level (JCI). MDELAY also receives the value of the minimum priority level to be delayed (JP2) and the number of messages not to be deleted

from that priority level (JC2). All messages of priority level less than the value of JP1 are sent (a value of 1 being the highest priority). At the JP1 priority level JC1 messages will be sent and the rest will be held. All messages at levels greater than JP1 and less than JP2 will be held. At the JP2 priority level JC2 messages will be held and the rest will be deleted. All messages above level JP2 will be deleted.

B.2.c(2)(p). Subroutine MINLIM

Subroutine MINLIM limits the number of messages that can be input in one time increment. If there are more input messsages than can be processed in one time increment then excess input messages are temporarily moved from the input message queue to the hold queue. After the messages that are still in the input message queue have been processed subroutine MINLIM is called again and the input messages are moved back from the hold queue to the input message queue so they can be processed next time. If the maximum number of input messages that can be procesed cuts off the input message queue in the middle of a priority level then all messages of that priority level are received.

B.2.c(2)(q). Subroutine Moulim

Subroutine MOULIM limits the generation of output messages at each node based on the number of messages created to be processed at each node and message priority. If there are more messages than can be output in one time increment then the appropriate number of messages (starting with the lowest priority) are either held (delayed one time increment) or deleted.

B.2.c(2)(r). Subroutine MovMsg

This utility is used by SUBROUTINE ALTOUT to attempt to allocate messages to alternative destinations. MOVMSG will

attempt to move messages in a queue with its root at PMSGO (in a link structure) and the first message located at PMSG in DM. It will try the first or second alternate destination in each message based on the value of IALT. The unit identified in IUNIT is checked to see if this is an acceptable alternate for a message. If it is acceptable, then the links that are available to go to this destination are checked to see if they are acceptable links for the message. For a link that is acceptable, the message is placed on that link if it has sufficient priority over existing messages already on the send queue and if it does not exceed the available link capacity. The alternate communications flag is set in the message to zero for primary link type and 1 or 2 for alternate link types.

B.2.c(2)(s). Subroutine MsgIn

Each node in the network has the capability to process messages. The data structure that holds the information about this process is the "output message process." The basic assumption for modeling the processing of messages that have been received is that received messages can be grouped together by message type and message originator unit type. MSGIN compares the input message to the message type and originating unit type for each message processed at the node. When matches are found, the process' "input message type" substructure, the "input message list" (IML), is searched for the specific unit identifier. If it is not found, an IML structure is created for the new unit identifier. In either case (existent or previous non-existent IML) the time the message was created is compared to the time of message creation in the IML. If the incoming message is newer, the message flag is set to 1 and message age is set to 0 in the IML. In addition, the IML message creation time is updated.

After all processes have been updated by the incoming message, it is tested to see if it contains an ATO substructure. If it does, the ATO is moved to the nodes ATO queue. Finally the data block containing the message is released to DM for reuse and control is returned to SUBROUTINE NODE.

B.2.c(2)(t). Subroutine MsgOut

SUBROUTINE PROCESS calls MSGOUT for each instance that a node has met the requirements to satisfy a message process. The functions performed are a determination of the desired destinations, alternate destinations, and message structure creation. The data structure used is Output Message Process (OMP) and its queue of output messages.

Each output message in the OMP is checked to see if it is addresssed to the commander of the node or if it is an air request. If the commander flag in the message is on (=1), the message is sent to the commander only. If it is not commander only or an air request, the message is sent to all units (nodes) that can be communicated with directly (in the DEST queue) that match the destination unit type found in the 5th word of the output message.

Next a message data block is located in DM and filled in with the data in the output message queue. The process for alternate destinations matches the unit types specified in the output message to the designated alternate units for the destination of the message. Finally, the message is placed on the node's send queue, by priority, or on its future message queue, by time to be sent.

MsgOut also has the ability to randomly vary the amount of communications capacity required to send a message. If option flag 19 equals 1 message, length will be modified for all messages of node type "300."

B.2.c(2)(u). Subroutine PRatio

Routine AtoAlo calls PRatio to calculate the commanders perceived force ratio for his subordinate units. Each of the eleven weapon system types for the foe is summarized over each of the foe units in the foe queue. The numbers of each system type is based on the generic unit strengths minus the number of foe weapons reported as destroyed in the status structure. The number of subordinate unit systems is taken from the status structure. Each weapon system ratio is calculated. The current unit force ratio for allocation of CAS sorties is the sum of red type one plus 4 times type 11 divided by the sum of blue type one plus 4 times type 11. PRatio also sets the force ratio calculation time to the current game time.

B.2.c(2)(v). Subroutine Process

Routine Node calls Proces to perform response actions based on the conditions of process rules having been met. When node determines that the conditions have been met for a rule number, Proces bases its actions on the output message types for the rule. It currently processes message types 2900, 3000, 3400, 7000, 9990, 9993, 3126, 3136 and 3800. Proces checks the first message type and branches to the related CASE statements. After each set of CASE statements control branches to increment the output message type queue. When all output message types for the process have been completed, routine AtoDel is called to delete all additional data structures that were obtained from input messages under this process number.

B.2.c(2)(w). Subroutine RanMsg

Routine Proces calls RanMsg in response to output message type 3800, 4800, 5800, 5900, 6800, 7800 and 7900. RanMsg checks itime to determine if it is the start of the game. If it is,

the node's random message queue is initialized with the processes random message using the start time in the generic message at MSG(15) plus 2 times that number times a uniform random number. Each message on the random message queue is checked to see if its send time has occurred. If it has, the message is scheduled again using the same calculation as above and MsgOut is called to create the actual message to be sent.

B.2.c(2)(x). Subroutine Send

SEND is the final subroutine in message processsing. It takes the contents of each link's send queue for each node and moves the entire queue as a complete string to the destination nodes input queue. The resulting sequence on a node's input queue is by sending node and by priority of the messages within each sender's segment. Each message sent by the node is counted and the running summation is stored in the NODE block. Then SUB-ROUTINE SEND checks each message on each hold queue to see if the message is overdue. If it is overdue, then that message is de-leted from the network and a notification is printed on output.

B.2.c(2)(y). Subroutine StatUp

Routine Proces calls StatUp in response to message types 3126 and 3130. These messages are created by routine RptLos at each combat cycle and sent through the network to the commander's node (or input from external messages). StatUp gets each data element from the node's spot report queue, and checks the process number in the report. Each applicable report with blue data flag set to 1 is used to update the subordinate's losses and strength estimates and blue's combat posture. If the blue data flag is off (equals o), StatUp updates the estimate of red forces des-troyed.

B.2.c(3). Air Operations Sub-module

The air operations module receives requests for CAS from nodes representing ground unit command posts (usually at the Corps level) in accordance with user established decisions rules. The reception of CAS requests and the resulting allocation, assignment, and scheduling of aircraft to support the requests are modeled at the wing operations level (WOC). The structure of air force combat resources starts with a CRC that controls airborne CAS for a designated set of ground combat nodes. The aircraft that the CRC controls come from one or more notional airbases that have a direct relationship to the CRC. Each airbase (WOC) may have one or more types of aircraft that are scheduled for sorties. A queue exists for each aircraft type at each WOC. This queue contains the number of aircraft that will be available for assignment at a specified time. The aircraft combat cycle starts in the availability queue, includes assignment, takeoff, reporting into the CRC, enroute to target, combat attrition (in the suported ground units combat matrix), aircraft survivability, return to airbase, turn-around for another mission, and back into the availability queue.

Requests for CAS are originated by the combat level units when their force ratios reach a user designated level. They may also be originated via EXTMSG input to a node. The requests for CAS are processed through the C^3 network and command posts in the same way all messages are handled with the exception that a commander may approve requests in accordance with an allocation plan. The WOC is a designated node type that has WOC processing capabilities and resources. Requests for CAS arrive at the NODE message structure for a WOC in the same way that all messages arrive at a node (in the NODE's INPUT queue). Notification of action on a CAS request is returned in the same manner. This module consists of four subroutines: AIROPS, MAKMSG, STATUS, and PRTATO.

B.2.c(3). Subroutine AIROPS

Aircraft resources on the ground at the notional airbase are maintained in aircraft type queues as shown in Figure 14. The first action by this subroutine is to determine the number of aircraft at the current game time. In Figure 14, if time is 3, the program would add 3.93 to the previously available 4.0 aircraft and then delete the RDYQ blocks with time 3. This is done for all aircraft types (READY BLOCKS). The aircraft availability status is printed out if requested by user input. Mission takeoffs are scheduled by starting with the requested Air Tasking Order (ATO) queue in the WOC's NODE structure. The projected time on target is calculated using alert time and en-route time and compared to the first request's earliest

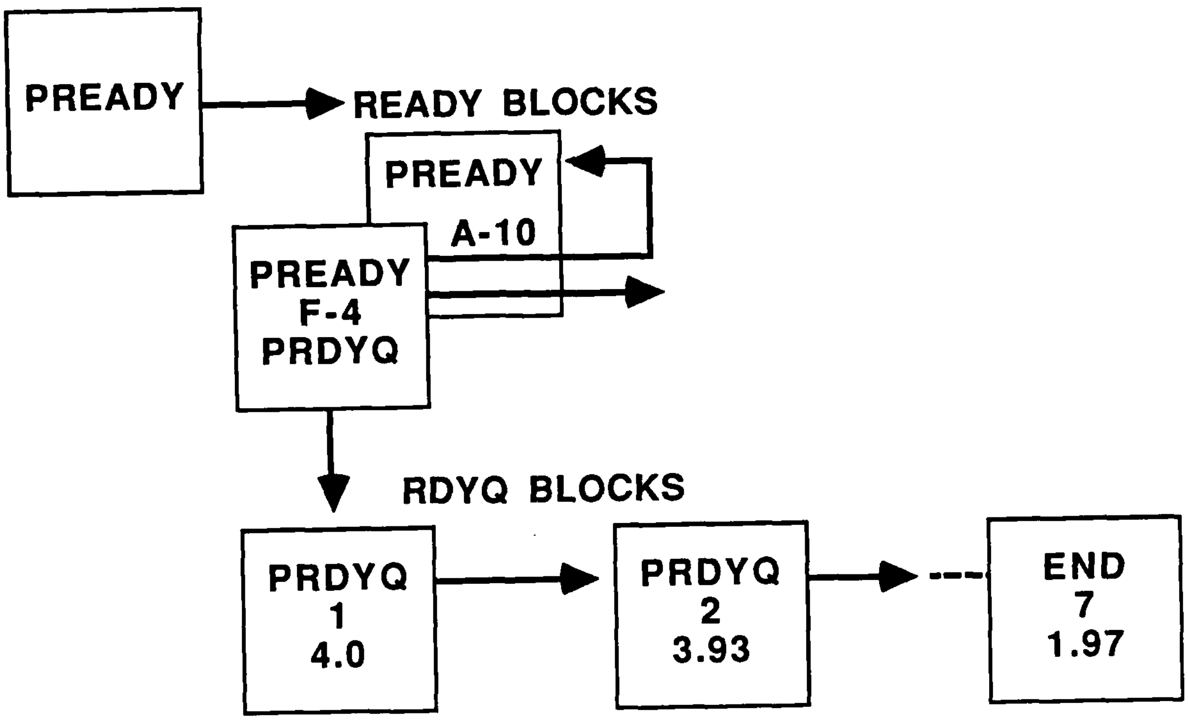


Figure 14: AIRCRAFT AVAILABILITY STRUCTURE

acceptable time on target (TOT). If the aircraft would arrive too soon, processing of air requests is finished for this WOC during this time frame. (Requests on this queue are sorted in earliest TOT order.) If the earliest time is acceptable, then the latest time is checked. If the mission would be too late, the request is deleted and a message is sent to the requesting node. If the mission can meet the requested window, processing is continued by checking the availability of the aircraft type requested and the number to be sent. All missions will have a whole number of aircraft assigned that is not less than the minimum aircraft limit specified by input. If the number requested is available, then the request is completely filled. If the number available is less than requested, the mission is scheduled with the reduced number.

If a request is fulfilled, messages to the CRC and requesting nodes are created by calling SUBROUTINE MAKMSG and placed on the WOC NODE's future message queue. An ATO is placed on the CRC's ATO queue. This queue is used to model the mission reporting into the CRC after takeoff. After processing all current requests for the WOC, the remaining aircraft availability status is printed out. SUBROUTINE AIROPS also processes CRC actions. The first step is to zero out all of the CAS entries in the combat matrix for each combat level unit. (Note that this means that time on target is always one time cycle.) Then the ATO queue is examined for each mission that is on target during this time interval. The combat unit's combat matrix is found and CAS is incremented by the number of aircraft in the mission. Also losses of CAS aircraft due to enemy action are computed. A running summation of CAS sorties is also kept. The last step is to calculate the number of aircraft that survive the mission. An enroute survivability is factored in and the total returning aircraft is computed. Note that this may produce fractions of an

aircraft. This number is then scheduled for landing and ground turn-around by entering it in the WOC's RDYQ queue.

B.2.c(3)(b). Subroutine MakMsg

This routine is called by Subroutine AirOps to create messages type 7000 that notify a combat unit of the time on target and number of aircraft that have been dispatched for CAS. This routine is documented in Section 3.2.c.2(n).

B.2.c(3)(c). Subroutine Status

This routine is called by Subroutine AirOps to print the status of a WOC on file C^3DATA.

B.2.c(4). Combat Sub-module

This module uses the IDA method of attrition calculation documented in IDA Paper P-1615, Net Assessment Methodologies and Critical Data Elements for Strategic and Theater Force Comparisons for Total Force Capability Assessment (TFCA), Volume II: Illustrative Example of Static Measures and Methodology. The executive routine for the attrition calculation is Subroutine Map. It is called by Subroutine Combat which interfaces betwen the C^3EVAL processing and data structures and the Map algorithms. This sub-module generates requests for CAS, determines aircraft losses during the attack portion of their mission, saves weapon system losses on file LOSST, and creates spot loss reports.

B.2.c(4)(a). Subroutine CasLos

Routine AirOps calculates the time on target and the time to return to the ready queue at the WOC for each CAS mission. It also calculates the enroute losses and schedules the remaining aircraft for return to duty by putting the returning aircraft on the WOC's ready queue. Attrition of CAS due to hostile systems

in the ground support area is calculated by routine Combat. Combat calls CasLos to add the additional loses to the mission aircraft. CasLos searches the WOC's ready queue for the first mission with return to ready time that is the same as the aircraft returning from combat and subtracts the combat losses from the number of aircraft to be available.

B.2.c(4)(b). Subroutine Combat

This routine is the executive for determining combat attrition. Combat interactions are evaluated each time increment for each unit that is a combat type unit that is not in combat status equal 0 (i.e., in reserve). Combat system and posture data is extracted from a node's combat structure for both Blue and the opposing Red force. The allocation matrices and force ratios are calculated by Subroutine Map. The combat drawdown is calculated by the matrix multiplication of the number of systems times the opposing sides Qmatrix created by Map. The results of the drawdown are stored in the node's combat structure and losses are output to file LOSST if the user has requested this data. File LOSST is used by the post processor to generate weapon system loss graphics. Subroutine Combat compares the current force ratio to the input threshhold level and creates a request message for CAS if the ratio is too high and earlier requests are not pending. If the user requested it, the results of each combat engagement is printed on file C^3DATA. If the user has indicated random processes are desired, Combat will randomly modify the times that requests for CAS are sent. Subroutine Combat calls RptLos to generate spot loss reports to his commander.

B.2.c(4)(c). Subroutine MakMsg

Subroutine Combat calls MakMsg to create messages to request CAS support. This routine is documented in Section B.2.c(2)(n).

B.2.c(4)(d). Subroutine Map

This routine is the executive for calculation of force ratios, weapon system allocations and the Q combat matrix. This routine and its subordinates are documented as noted in (4) above.

B.2.c(4)(e). Subroutine RptLos

This routine is called by Combat for each unit after combat attrition is calculated. It creates two combat spot loss reports, message number 3130 for Blue and 3126 for Red losses. These reports are sent to the unit's commander and forms the basis for the commander's perceptions of the unit's status. The actual losses are reported unless the random process flag is on. Under random operations, the messages are randomly delayed and the contents of the messages are randomly modified.

B.2.d. Output Module

Output by C^3EVAL is provided in four different areas; input echo, game events, summary and time T. Input data is echoed to file C^3ECHO to assist in creating a complete record of the scenario and to verify that the data structures created during initialization are properly filled. Subroutine Input echos the data preamble, control flags, node data set, link data set, and limits data set. Subroutine Inputs echos combat unit strengths. Subroutine Inputa echos CAS allocation parameters, CRC parameters and available aircraft. Subroutine RulPrt prints out the decision rules in node type (echelon) order.

Game events are printed to file C^3TIME. Subroutine StatOn prints out the commander's initial perceptions of his subordinates. TimeT input of reinforcements, combat status and link capacity changes are printed out by routines under TinPut. ExtMsg echos the user's input messages as they occur. The

printout of message flows, queue status, combat losses and scheduling of CAS is controlled by user flags set by routine Contrl. If the debug flag is set, a large volume of physical as well as logical data is printed on file C^3DATA.

At the end of the game, a summary of communications, network and support sorties are printed to file C^3SUM. This is followed by the size of dynamic memory actually used and the commander's perceptions at game end.

The fourth area of output is created for use by the graphic post processor. The message flow data at each time interval may be printed to file TIMET and the combat losses data printed to file LOSST.

B.2.d(1). Subroutine OutPut

Subroutine Events calls OutPut if the user has requested time-T data to be saved, PFLAG(14), for the post processor. Events set the output file, IOUT to TIMET calls OutPut, and then resets IOUT to C^3DATA afterward. The main program C^3EVAL calls OutPut at the end of the game with IOUT set to C^3SUM. OutPut creates the printout shown in Figure 15.

B.2.d(2). Subroutine PMSG1

This routine prints out data about messages at alternate destinations and on a node's input and future message queues. It may print all messages, all messages at a specified node, and/or during a specified time frame. It may be restricted to only those messages that have their track flag set.

B.2.d(3). Subroutine PMSG2

This routine processes the same as PMSG1 except that it is done for a node's output and hold queues.

SUMMARY OUTPUT AT TIME 48

			COMMUNICATIONS LIMIT				INPUT LIMIT			OUTPUT LIMIT			SORTIES	
UNIT	NUMBER	TYPE	IN	OUT	HOLD	KILL	IN	HOLD	KILL	OUT	HOLD	KILL	CAS	HELO
SHAPE	18	700	43	13	0	0	43	0	0	13	0	0	0	0
AFCENT/AAFC	17	600	185	134	0	1	185	0	0	135	0	0	0	0
VII CORPS	16	400	7	2	0	0	7	0	0	2	0	0	0	0
VII CORP TA	15	450	28	5	0	0	28	0	0	5	0	0	0	0
CENTAG	14	500	94	54	0	0	94	0	0	54	0	0	0	0
V CORP REAR	13	490	43	24	0	0	43	0	0	24	0	0	0	0
V CORPS TAC	12	450	437	69	0	3	437	0	0	72	0	0	0	0
52 MECH	9	300	78	273	24	5	58	0	0	274	0	0	0	0
WOC	8	7000	13	105	0	0	13	0	0	105	0	0	0	0
ATOC	7	5000	108	224	0	0	108	0	0	224	0	0	0	0
4ATAF	6	6000	249	173	0	0	249	0	0	173	0	0	0	0
23 ARM DIV	4	300	93	174	2	1	62	20	11	177	54	47	54	0
V CORPS	3	400	173	102	0	0	120	0	0	49	0	0	0	0
20 MECH	2	300	57	169	0	0	57	0	0	169	0	0	0	0
201 ACR	1	250	23	14	0	0	18	0	0	9	0	0	0	0

Figure 15: SUMMARY OUTPUT AT TIME

B.2.d(4). **Subroutine RulOut**

Routine Events calls RulOut if PFLAG(7) is set. The main program C^3EVAL calls RulOut if PFLAG(17) is set. RulOut prints out the decision rule parameters and variables in physical structure form so that the details of their operation can be followed. If the debug flag is on RulOut will call PMsg1 and PMsg2.

B.2.d(5). **Subroutine RulPrt**

The main program C^3EVAL calls RulPrt to echo the decision rules to file C^3ECHO. RulPrt loops through each decision rule for each level unit specified in its internal data statement. If new unit types are added to the scenario, the unit type must be added to this data statement to have its rules echoed to the file. The decision rules are printed in unit type order with the rule process parameters followed by the required message-in data and then the output messages to be generated by the rule.

3.2.d(6). **Subroutine StatOu**

Routine Events calls StatOu at the start of event processing to print the commander's initial perceptions of his subordinates. The main program C^3EVAL calls StatOu at the end of the game to print the commander's perceptions at the end. StatOu checks all nodes to determine if their status queue exists. If it does, the integer value of blue and red curent perceived forces and losses are printed. Blue data and red losses come directly from the status structure. Red perceived weapon system numbers are obtained by adding the values for each foe perceived from the generic tables and subtracting the losses reported in the status structure. The perceived combat posture is also printed.

3.2.d(7). Graphics Data

Summary data is printed by subroutine Output. If the user has indicated graphics post processor data is desired, routine Events calls Output at each time interval. This data is written in character format to file TIMET. The post processsor reads this data to create some of its graphic output options. In addition, if the graphics data flag is on, routine Combat writes the comat losses and force ratio for each node in combat. This data is written in binary form to file LOSST. The post processor also reads this file.

B.2.d(8). Subroutine VMData

This routine prints out the maximum dynamic memory location used and the status of the reusable block queues.

B.2.e. Utilities

These routines perform data structure building, searches, dynamic memory operations, and other program support-type functions.

B.2.3(1). Subroutine Find

SUBROUTINE FIND searches the elements of a queue for an input calling sequence integer value (ID). The starting point in the queue is PIN. FIND assumes that the pointer to the next element in the queue is the first value in an element and that the last element in the queue will have this value set to zero. The offset from the first value of an element that contains the desired value is indicated by the parameter N. If the value is not found, the output parament POUT will be set to zero.

B.2.e(2). Subroutine Gimme

GIMME provides a DM data block of length LEN from the memory space ISPACE. In the current version of C^3EVAL, there is only one dynamic memory area (MEMORY). The location of the first word in the data block is set into NPTR. GIMME first searches its garbage list to see if a block of equal length is available for reuse. If not, it creates a new block of the desired length by increasing next unused space pointer ISPTR by the value of LEN. GIMME also checks to insure that the current maximum size for DM is not exceeded.

B.2.e(3). Subroutine POut

This routine is used for debug purposes only. It produces a snapshot of part of dynamic memory. The snapshot starts on location one and prints the specified number of variables (up to eleven) per line and the specified number of lines. Note that C^3EVAL does not use locations 1-100 and this area should be all zeros.

B.2.e(4). Subroutine Releas

This routine works in conjunction with Gimme to control dynamic memory. It places data blocks on the reusable memory queue. The block address is placed on a queue of blocks that have the same length as the input parameter to Releas. The queue has its root in variable IGBPTR. Note that there is no garbage collection accomplished.

B.2.e(5). Subroutine Restor

Restore is used to restart a game at time T other than zero. It assumes that a previous run has been made and that the status of dynamic memory and COMMON parameters were saved by routine Save. This capability has not been enhanced to operate with the

current version of C^3EVAL and will not operate correctly until it has the ability to handle the time T input files created in the current version.

B.2.e(6). Subroutine Save

This subroutine saves the status of DM and model parameters. This data may be used to restart the game at the point where SUBROUTINE SAVE was called. Due to enhancements made to provide addditional time T inputs, this subroutine requires enhancement before it can be used with the current version.

B.2.e(7). Subroutine SNAP

SUBROUTINE SNAP finds the correct position in a queue with root in parameter PTR to insert a new member located at PIN. The sequence variable is located at NWORD in the data structures.

B.2.f. Data Structures

There are two distinctly different approaches to data structures used in C^3EVAL. The use of FORTRAN common variables and arrays is described first. The next section defines the approach used to provide essentially dimensionless code and the linked list data blocks that are used to implement the required data structures.

B.2.f(1). Common Data Structures

The following nine named common data structures are used in C^3EVAL. This implementation is standard FORTRAN, with the exception of variables that start with "P" which are declared implicit integer variables. They are normally used to represent pointer (locators) of linked list data blocks in C^3EVAL's dynamic memory.

(a.) COMMON/C3/NODE1,PGOMT,NCRC,INP,IOUT,NREDTE,IRAND

NODE1	Location of first node data structure in DM
PGOMT	Not currently used
NCRC	Location of first CRC data structure in Dynamic Memory
INP	Input file number of for C^3 data, set to 1 in PROGRAM C^3EVAL
IOUT	Output file number of all output, set to 6,7 and 8 in PROGRAM C^3EVAL and 14 in Events
NREDTE	Location of first generic red unit table of equipment
IRAND	Seed for random number generator set to 731593 in Block Data FRatio

The node data block is the basic building block for all C^3 and combat data. The descriptions of the node, CRC and red generic data blocks are in Section (2).

(b.) COMMON/SPACE/NOUSE,MEMORY (20000).

NOUSE	A check variable used for debug purposes only. (It is MEMORY location zero).
MEMORY	DM array. It is equivalenced to STORE to facilitate its use for floating point as well as integer values.

(c.) COMMON/LOCATE/ISPTR,IGBPTR,MAXSP.

ISPTR	Location
IGBPTR	Root of the linked list of reusable data blocks in DM
MAXSP	Maximum value for ISPTR

(d.) COMMON/TIME/ITIME,INCTIM,LASTT,PD1,PD2

ITIME	Current game time, number of INCTIM intervals that have been simulated.
INCTIM	Basic time interval of model, set to 1 in C^3EVAL meaning one 30-minute period.
LASTT	Last time for this simulation run (for 24 hours of combat LASTT=48).

PD1 Start time for debug print (if PDEBUG=0)

PD2 Stop time for debug print

(e.) COMMON/RED/and COMMON/BLUE/.

```
      COMMON/RED/ NR(11),ALTCRB(11,11),ER(11),VALR(11),
     1            PKRB(11,11),NTCR(11),QR(11,11),ALLRB(11,11),
     2            NTRJ,WGTR(11),WR(11)
      COMMON/BLUE/NB(11),ALTCBR(11,11),EB(11),VALB(11),
     1            PKBR(11,11),NTCB(11),QB(11,11),ALLBR(11,11)
     2            NTBI,WGTB(11),WB(11),I1
```

With the exception of I1 in COMMON/BLUE/ these two commons are identical in definition for Red and Blue forces. The Blue definitions will be given here.

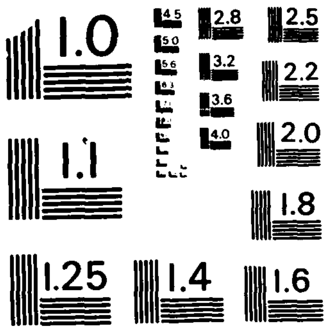
NB	Number of weapons by type
ALTCBR	Typical allocation of Blue weapons against Red
EB	Engagement rate for each Blue weapon system tions
VALB	Interface variable for other APP calculations
PKBR	Probability that a Blue system kills a Red system
NTCB	Typical number of Blue weapons assigned
QB	Kill rate matrix of Blue against Red
ALLBR	Allocation matrix of Blue against Red
NTBI	Number of types of weapons, set to 11
I1	Index weapon system
WGTB,WB	Interface variables for other APP calculations

(f.) COMMON/EIGEN/EIGR(11),EIGB(11),V(11),BETA,RAT4

EIGR	Force ratio eigen vector for Red
EIGB	Force ratio eigen vector for Blue
V	Initial guess vector for eigen solution
BETA	1/lamda of eigen matrix
RAT4	Force ratio Red to Blue

AD-A168 973 C3 EVAL MODEL DEVELOPMENT AND TEST VOLUME 2 PROGRAMMERS MANUAL..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA VA R F ROBINSON ET AL. OCT 85 IDA-P-1882-VOL-2 2/3

UNCLASSIFIED IDA/HQ-85-30596 MDA903-84-C-0031 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(g.) COMMON/ENGMT/ERP(3,11),EBP(3,11),POSR,POSB,LCMBT, FRBCAS.

ERP	Engagement rate for Red
EBP	Engagement rate for Blue
POSR	Combat posture for Red
POSB	Combat posture for Blue
LCMBT	Unit type identification of combat units
FRBCAS	Force ratio Red to Blue limit for CAS requests

(h.) COMMON/NAMES/NAMER(11),NAMEB(11).

NAMER	Names of Red combat systems
NAMEB	Names of Blue combat systems

(i.) COMMON/PRTFLG/PFLAG(20),PMOD(3),PDEBUG

PFLAG	Array of user control flags, see Figure 8.
PMOD(1)	Message printout for this node only, if zero do all nodes.
PMOD(2)	Output start time
PMOD(3)	Output stop time
PDEBUG	Debug print flag, if 1 debugger is on

B.2.f(2) Dynamic Data Structure (DDS)

There are three sets of DDS in C^3EVAL. They are the NODE, CRC and GENERIC sets. A set has its root in a non-dynamic location and is linked together by pointers. As shown in Figures 16, 17, and 18, the root locations are variables NODE, NCRC and NREDTE in COMMON/C^3/. These structures are created during input by calls to SUBROUTINE GIMME which acquires data blocks from dynamic memory. The length of each type data block is a fixed number in the code. This section lists the data blocks, defines their elements, gives the type of each variable, identifies the routine that creates the block and the ones that delete the block if applicable, and specifies the location of the root.

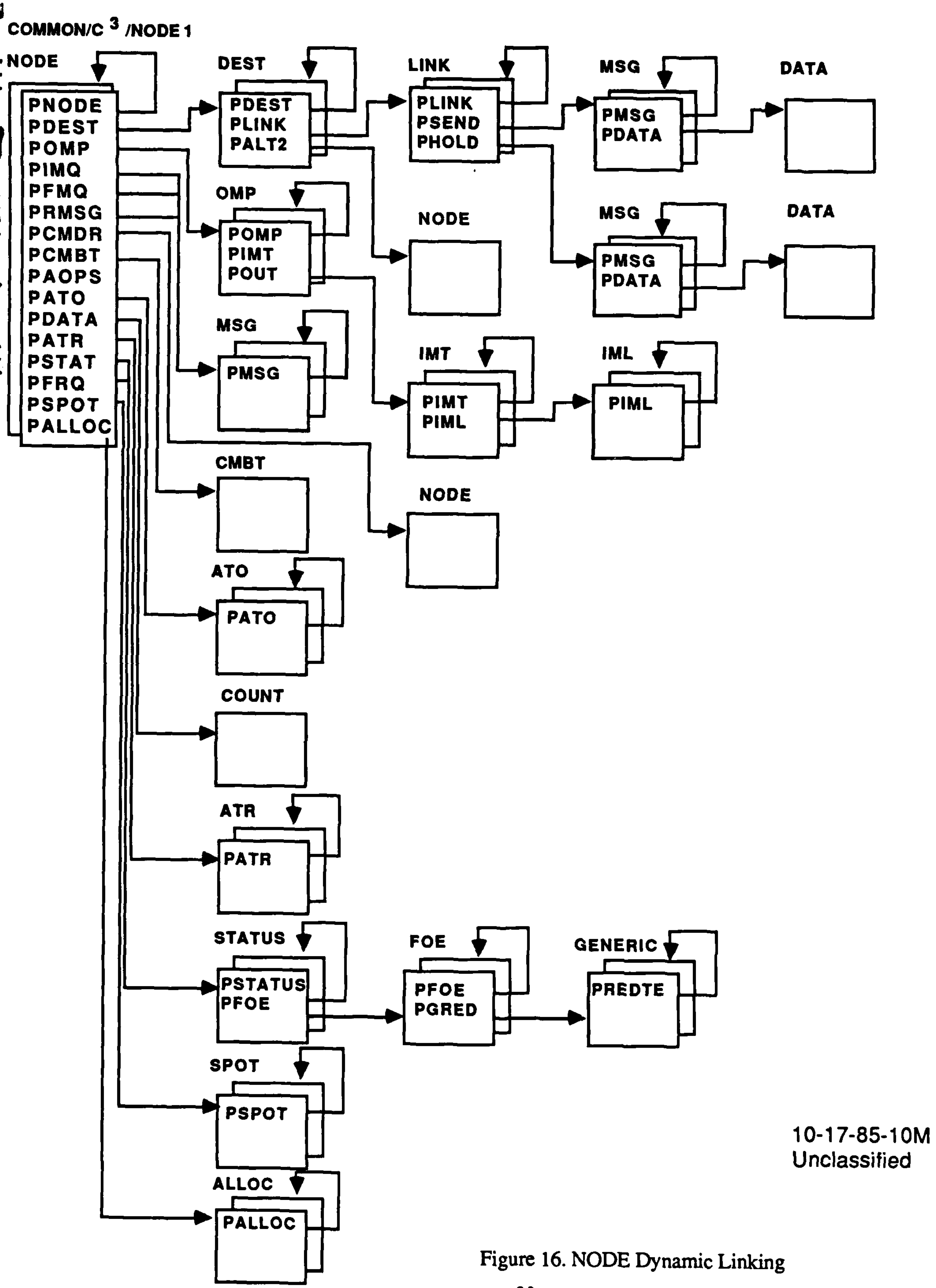


Figure 16. NODE Dynamic Linking

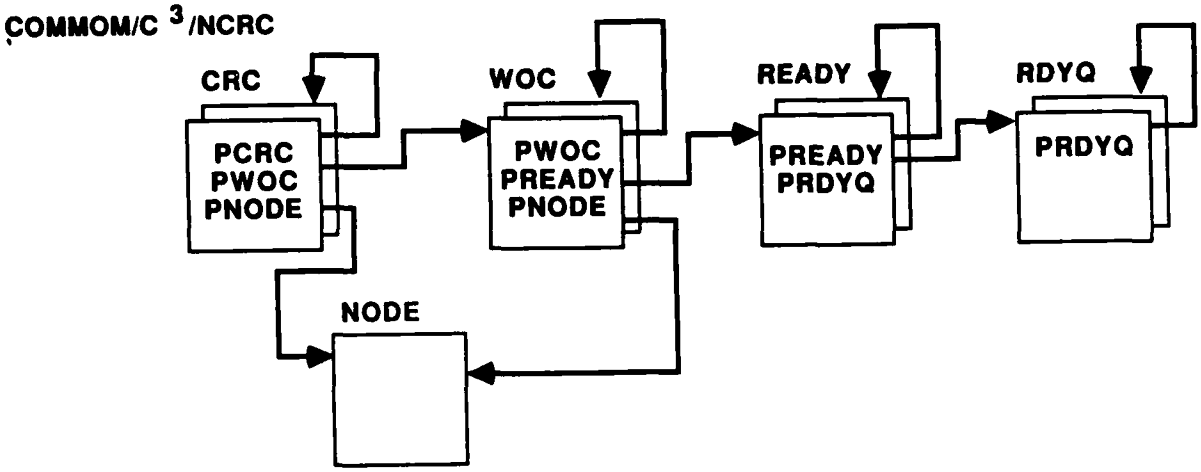


Figure 17: DYNAMIC DATA LINKING

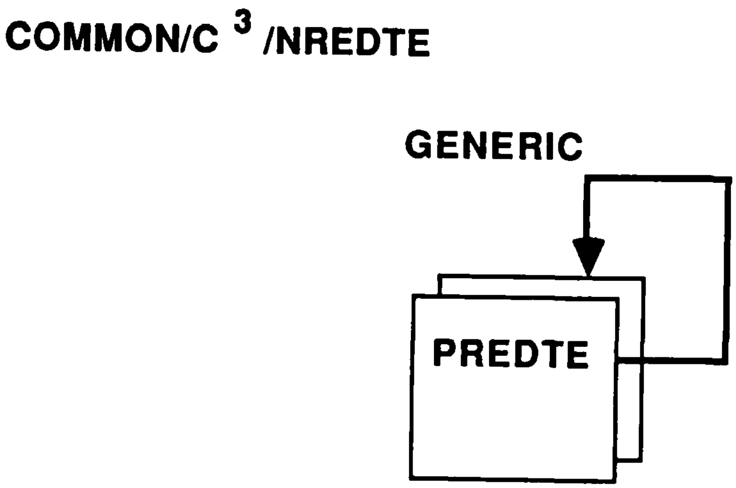


Figure 18: GENERIC RED UNIT TABLES OF EQUIPMENT
DYNAMIC DATA LINKING

The symbols used in the DDS's documented in this section are:

SYMBOL	TYPE OF ELEMENT
P	Pointer to another DDS
I	Integer value
R	Real value
C	Character

The DDS is traversed by the links shown in Figures 16-18. While each node is of a fixed length, the number of DDSs in a single queue is unlimited. Therefore, one node may have only one destination while another may have several.

When a specific data element is desired, the program code uses successive pointers to locate that element.

Example: Find the capacity of communication link type 16 that connects node 3 to node 4.

1) Find node 3 by starting at COMMON/C^3/NODE1. This is done in code by setting PNODE=NODE1. This is a location in DM. From the NODE DDS below, we see that location PNODE contains a pointer to the next NODE structure and that location PNODE+1 contains the unit identifier. This identifier is compared to the node number desired (3). If this is not the desired NODE, then PNODE is reset to MEMORY(PNODE), which is the location of the next node structure. This series is repeated until the desired NODE is found. (This can be done by SUBROUTINE FIND.)

2) From the NODE DDS we see that PNODE+3 is a pointer to a communications destination queue. (The offset PNODE+3 is always one less than the number in the attachment, because PNODE is the location of the first element in the DDS.) The DEST (destination) structure is traversed to find unit identifier 4 in the same manner that was used in step 1.

3) The DEST DDS contains a pointer to its communications links. These links are traversed until a LINK DDS with type 16 is found.

4) The capacity of this link is located at PLINK+2.

BLOCK NAME:	ALLOC	BLOCK SIZE:	7
USE:	This structure contains the allocation parameters for a commanding unit to use in approving requests for CAS by its subordinates.		
CREATED BY:	INPUTA		
DELETED BY:	N/A		
ROOT:	NODE(21)	DATE:	

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PALLOC	P	Pointer to next allocation structure
2	TYPE	I	Type of allocation data (3000,3400)
3	SAT	I	Start allocation time
4	NAT	I	Number of allocation time periods
5	NSOR	I	Number of sorties allocated
6	MINSOR	I	Minimum sorties on a mission
7	MAXSOR	I	Maximum sorties to be allocated

BLOCK NAME: ATO — BLOCK SIZE: 15

USE: **Contains the data portion of a request for CAS or the response to a CAS request.**

CREATED BY: **MSGOUT**

DELETED BY: **SEND**

ROOT: **PMSG(18)** — DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PATO	P	Next air task order
2	ID	I	Support unit number
3	ETIME	I	Earliest support time OR takeoff time
4	LTIME	I	Latest support time OR on target time
5	ACTYPE	I	Type of aircraft

BLOCK NAME:	CMBT	BLOCK SIZE: 62
USE:	**The combat data structure contains the combat weapon systems data for each combat level unit.**	
CREATED BY:	**INPUTC**	
DELETED BY:	**not applicable**	
ROOT:	**NODE (9)**	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	CFLAG	I	=0 no combat, =n > 0 n is period of combat, =m > 0 m is period of force ratio calculation
2	RATR	R	Blue ratio
3	RATR	R	Red ratio
4	RAT4	R	Force ratio Reb/Blue
5	BETA	R	Eigen value of 1/lamda
6-16	NB	R	Number of Blue weapons
17-27	NR	R	Number of Red weapons
28-39	EIGB	R	Eigen vector Blue
39-49	EIGR	R	Eigen vector Red
50-60	V	R	Eigen best initial guess
61	POSB	I	Posture of Blue: =0 not in combat, =1 Low, =2 Medium = 3 High Intensity
62	POSR	I	Posture of Red: =0 not in combat, =1 Low, =2 medium = High Intensity

BLOCK NAME:	**COUNT**	BLOCK SIZE: 30
USE:	**This structure contains the message counters, limits and other parameters held for output at each node.**	
CREATED BY:	**INPUT**	
DELETED BY:	**N/A**	
ROOT:	**NODE (12)**	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	CIN	I	Number messages arrive on network
2	COUT	I	Number messages sent out on network
3	CHOLD	I	Number messages held by comm. limit
4	CKILL	I	Input limit to hold messages
5	LINH	I	Input limit to hold messages
6	LINK	I	Input limit to kill messages
7	IIN	I	Number messages arrive thru limit
8	IHOLD	I	Number messages held by input limit
9	IKILL	I	Number messages killed by input limit
10	LOUH	I	Output limit to hold messages
11	LOUK	I	Output limit to kill messages
12	OIN	I	Number messages sent out thru limit
13	OHOLD	I	Number messages held by output limit
14	OKILL	I	Number messages killed by output limit
15			
16	CAS	I	Number CAS sorties in combat
17			
18	HELO	I	Number helicopter sorties in combat
19-30			

BLOCK NAME:	CRC	BLOCK SIZE: 5
USE:	Contains the parameters for a combat reporting center and the relationship to its node structure and wing operations support unit.	
CREATED BY:	INPUT	
DELETED BY:	not applicable	
ROOT:	/C^3/NCRC	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PCRC	P	Next CRC
2	ID	I	Unit number of CRC
3	TYPE	I	Unit type of CRC
4	PNODE	P	Pointer, CRC's node structure
5	PWOC	P	Pointer, wing ops structure

BLOCK NAME:	DEST	BLOCK SIZE: 9
USE:	This structure contains all of the other nodes which can be communicated with by the node to which it is attached.	
CREATED BY:	INPUT	
DELETED BY:	N/A	
ROOT:	NODE(4)	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PDEST	P	Pointer to next destination element
2	ID	I	Destination unit identifier
3	PLINK	P	Pointer to communications queue
4	PALTI	P	Pointer to first alternate
5	PALT2	I	Pointer to second alternate
6	TYPE	I	Destination unit type
7	ALOFLG	I	Allocation flag, =1 allocated
8			Internal flag
9	SUBFLG	I	Subordinate flag =1 subordinate

BLOCK NAME:	**FOE**		BLOCK SIZE: **7**
USE:	**Identifies specific red unit as foe of a subordinate and points to the generic red unit table of equipment (TE).**		
CREATED BY:	**STATIN, INTLUP**		
DELETED BY:	**INTLUP**		
ROOT:	**STATUS(27)**		DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PREDTE	P	Pointer to next FOE structure
2	UTYPE	I	Type of red unit
3-5	UNITID	C	Red unit name
6	PGRED	P	Pointer to generic FOE
7	TIME	I	Time of unit identification

BLOCK NAME:	GENERIC	BLOCK SIZE: 15
USE:	Structure for table of equipment for a generic Red unit.	
CREATED BY:	INPUTC	
DELETED BY:	N/A	
ROOT:	/C^3/NREDTE	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PREDTE	P	Pointer to next generic red TE
2	UTYPE	I	Type of Red unit
3-13	SYSTEM	R	Number of combat weapon systems
14	POSTUR	I	Combat posture

BLOCK NAME:	IML	BLOCK SIZE: 5
USE:	The input message list contains the variables that indicate specific message receipt from a designated usit (NODE).	
CREATED BY:	MSGIN	
DELETED BY:	not applicable	
ROOT:	IMT(4)	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PIML	P	Next input message list
2	NODE	I	Unit identifier of originator
3	MFLAG	I	Flag, message was received
4	MAGE	I	Age of last input message
5	OTIME	I	Time last input was originated

BLOCK NAME:	IMT	BLOCK SIZE:	6
USE:	**The input message type structure contains the parameters required to process a specific input mesage type that originated at a specific unit type. Each IMT has an IML queue.**		
CREATED BY:	**RULEIN**		
DELETED BY:	**not applicable**		
ROOT:	**OMP(3)**	DATE:	

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PIMT	P	Next input message type
2	MSG	I	Type of input message
3	OTYPE	I	Type unit that originates message
4	PIML	P	Input message list
5	MAXAGE	I	Maximum age of useful message
6	USE	I	Message use if 0 only once

BLOCK NAME: **INTEL REPORT** BLOCK SIZE: 21

USE: **Structure for foe identification message type 3136.**

CREATED BY: **INTLUP**

DELETED BY: **ATODEL, INTLUP**

ROOT: **MSG(18)** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PDATA	P	Pointer to next data report
2	ORIG	I	Node identifier of Blue combat unit
3	TIME	I	Time report data was created
4	TYPE	I	Type of foe unit
5	POSTUR	I	Posture of foe unit
6-8	NAME	C	Name of specific unit
9	PINTEL	P	Pointer to next INTEL report
10-16			Unused
17	PROCES	I	Input process number
18-21			Unused

BLOCK NAME: **LINK** BLOCK SIZE: 6

USE: **This structure contains the parameters for a link between the node and the destination in the root DEST structure.**

CREATED BY: **INPUT**

DELETED BY: **N/A**

ROOT: **DEST(3)** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PLINK	P	Pointer to next link element
2	LTYPE	I	Type of link
3	LCAP	I	Capacity of link
4	PSEND	P	Pointer to message send queue
5	PHOLD	P	Pointer to message hold queue
6	PWORK	P	Temporary queue for allocation

BLOCK NAME: MSG		BLOCK SIZE: 21
USE:	Basic message structure for all messages.	
CREATED BY:	ATORTN, EXTMSG, MAXMSG, MSGOUT, RULEIN,	
DELETED BY:	MDELAY, MINLIM, MSGIN, NODE, SEND,	
ROOT:	NODE(6), (7), (23); LINK(4), (5); IMT (4), OMP (4)	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PMSG	P	Pointer to next message element
2	STIME	I	Time to send messages
3	MTYPE	I	Type of message
4	GTYPE	I	Originator of message
5	DEST1	I	Destination uniut identifier
6	PRIOR	I	Priority of message
7	LTYPE1	I	Primary path type
8	CAP2	I	Capacity required on path 1
9	LTYPE2	I	First alternate path type
10	CAP2	I	Capacity required on path 2
11	LTYPE3	I	Last alternate type
12	CAP3	I	Capacity requried on path 3
13	DEST2	I	Alternate unit type
14	CMDR	I	=1 destination is commander only
15	FLAG1	I	Output flag
16	FLAG2	I	Alternate transmission flag
17	DEST3	I	Alternate unit type
18	PDATA	P	Pointer to additional data
19	NODE	I	Unit identifier of originator
20	OTIME	I	Time message was created
21	MAXAGE	I	Maximum time to be on the network

BLOCK NAME:	NODE		BLOCK SIZE: 23
USE:	**This structure is the top element for each node in the game. It is used to locate the functional structures that contain the node's characteristics.**		
CREATED BY:	**INPUT**		
DELETED BY:	**N/A**		
ROOT:	**/C^3/NODE1**		DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PNODE	P	Pointer to next node
2	ID	I	Unit identifier number
3	TYPE	I	Unit type
4	PDEST	P	Pointer to destination queue
5	POMP	P	Pointer to output message process
6	PIMQ	P	Pointer to input message queue
7	PFMQ	P	Pointer to future message queue
8	PCMDR	P	Pointer to commander
9	PCMBT	P	Pointer to combat data structure
10	PAOPS	P	Pointer to assigned air support
11	PATO	P	Pointer to air requests
12	PDATA	P	Pointer to counters for output
13-15	NAME	C	Unit name
16	PATR	P	Pointer to acknowledged requests
17	PFSO	P	Pointer to requested fire support
18	PFSR	P	Pointer to approved fire support
19	PSTAT	P	Pointer to subordinate status queue
20	PSPOT	P	Pointer to spot report queue
21	PALLOC	P	Pointer to allocation parameters
22	PFRQ	P	Pointer to force ratio queue
23	PRMSQ	P	Pointer to random message queue

BLOCK NAME:	OMP	BLOCK SIZE: 11
USE:	The output message process structure contains the parameters for message handling. Each OMP has an IMT queue and a MSG queue.	
CREATED BY:	RULEIN	
DELETED BY:	not applicable	
ROOT:	NODE(5)	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	POMP	P	Next output message type
2	RNO	I	Rule number
3	PIMT	P	Input message type queue
4	POUT	P	Output message queue
5	LTIME	I	Last time this process complete
6	OTYPE	I	Type unit that originated process
7	CTIME	I	Time to complete process
8	MFLAG	I	> 0 minimum current messages to do this process
9	TFLAG	I	=0 not period > 0 period interval
10	PRULES	P	Data element for action
11	STIME	I	Start time of periodic process

BLOCK NAME: RDYQ BLOCK SIZE: 3

USE: The aircraft availability queue contains the number of a specific aircraft type that will be available for takeoff at a specified time.

CREATED BY: AIROPS, INPUTA

DELETED BY: not applicable

ROOT: READY(3) DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PRDYQ	P	Next aircraft element
2	ACTIME	I	Time aircraft will be ready
3	NUMAC	I	Number aircraft to be ready

BLOCK NAME:	READY	BLOCK SIZE:	3
USE:	Identifies a specific aircraft type under control of the WOC		
CREATED BY:	INPUTA		
DELETED BY:	not applicable		
ROOT:	WOC(9)	DATE:	

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PREADY	P	Next aircraft element
2	ACTYPE	I	Aircraft type
3	PRDYQ	P	Aircraft ready queue

BLOCK NAME:	STATUS	BLOCK SIZE: 43
USE:	**Status of subordinate units, created if subordinate flag in input =2.**	
CREATED BY:	INPUT	
DELETED BY:	N/A	
ROOT:	NODE(19)	DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PSTATUS	P	Pointer to next status structure
2	UNITID	I	Identifier of status unit
3	TIMEB	I	Last time before status updated
4-14	NB	I	Number of blue weapons
15-25	BLOSS	R	Number of blue losses
26	TIMER	I	Last time red status updated
27	PFOE	P	Pointer to red foe structure
28-38	RLOSS	R	Number of red losses
39	POSB	I	Blue posture
40	POSR	I	Red posture
41	FRB	R	Red to Blue force ratio
42	PFRQ	P	Pointer to next status structure in force ratio queue
43	LFRT	I	Last time force ratio calculated

BLOCK NAME: **SPOT REPORT** BLOCK SIZE: 21

USE: **Structure for combat loss data for message types 3126 & 3130.**

CREATED BY: **RPTLOS**

DELETED BY: **ATODEL**

ROOT: **MSG(18) and NODE(20)** DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PSPOT	P	Pointer to next data report
2	ORIG	I	Node identifier to report originator
3	TIME	I	Time report was created
4	FOE	I	Flag; =0 foe, =1 subordinate data
5-15	LOSS	R	Losses of combat systems
16	POSTUR	I	Posture of unit
17	PROCES	I	Input process number
18-21			unused

BLOCK NAME: WOC BLOCK SIZE: 10

USE: This structure contains data on aircraft on the ground and parameters that effect their assignment to CAS missions

CREATED BY: INPUTA

DELETED BY: not applicable

ROOT: CRC(5) DATE:

INDEX	ELEMENT NAME	TYPE	ELEMENT MEANING/USE
1	PWOC	P	Next WOC
2	ID	I	Unit number of WOC
3	TYPE	I	Unit type of WOC
4	PNODE	P	WOC's node structure
5	ATIME	I	Alert response time
6	ETIME	I	Enroute time
7	MINAC	I	Minimum aircraft on mission
8	PS	R	Probability of survivability enroute
9	PREADY	P	Aircraft ready queue
10	RTIME	I	Ground turn-around time

B.2.g. Program Notes

This section contains notes on unique situations in the C^3EVAL model and its data bases.

During the development of the code it became convenient to preassign certain data values. These numbers may be required in the data sets and their use must coincide with the definitions below:

Node type for WOC	7000
Message number for external air requests	3000,2900
Message number for force ratio air requests	3400
Message number for CAS notifications	7000
Message number for red losses	3126
Message number for blue losses	3130
Message number for foe's identification	3136
Process numbers for random created messages	3800,4800,5800, 6800,7800,5900,7900

A message type 3400 is created in subroutine Combat when the force ratio falls below the input threshold value. The force ratio is a function of all 11 weapon types against the systems allocated. If CAS is zero then SAMS will contribute nothing to the force ratio. Adding a few blue CAS sorties into combat with a large number of SAMS with cause the force ratio to shift significantly to the side with SAMs.

Routine RulPrt has the types of nodes in a data statement. If additional types are used they will not be printed out on File C^3ECHO unless this array is changed.

The following comments pertain to input values. File C^3DATA has a data set "LMNO." The first value is the input threshold to hold messages that exceed this count. However, all messages input at the node that have the same priority are treated the same. For example, if the hold limit is 5 and there are 3

priority 1's, 3 priority 2's and 3 priority 3 messages, the algorithm will allow all 3 priority 1's and all 3 priority 2's to be input because the limit fell in the priority 2 count range. All messages with priorities higher than 2 will be held (if their data useful time is not exceeded) or deleted. The priority level algorithm is also applied to messages at the delete limit. The limits are cumulative (i.e., if the limits are 5 and 9) the number of candidates to be held is 4).

The third and fourth values are for output limits which operate the same as input except that the priority level algorithm is not applied.

Post processing graphics read files TIMET and LOSST to get their input. These files are created by C^3EVAL if the graphic flag is on in C^3DATA, Flag Number 14. Red and blue weapon strengths and combat postures may be changed during the game. File CBDATA would have two additional lines of data (1 for red and 1 for blue) with the desired times for the change. All 11 systems are modified by this input (a minus sign removes forces) with the exception of blue CAS. This field is not used. All CAS for blue must be requested through the network and is subejct to aircraft availability of the WOC.

Message types that are created by successful completion of a decision rule are used to create specific messages at a node and are placed on the message type's primary destination path and communications type to the receiving node. If the primary path and communications type does not exist (note capacity may be zero but the path exists) then the message will be deleted and an error written on C^3TIME.

The AODATA file may contain CAS allocation parameters for a node that commands combat level units. If allocation parameters exist for the type of CAS request message received, subroutine ATOALO approves the requests if they meet the allocation criteria

of number of sorties currently available to allocate and the priority of the request. This priority is based on the node's perception of the requestor's force ratio. This force ratio is calculated by routine PRATIO as the sum of red weapon types one plus 4 times eleven divided by the blue weapon types one plus 4 times eleven. Requests for CAS are filled completely for each request as long as sorties are available.

B.2.h. Internal Code Documentation

```
      PROGRAM C3EVAL
*****************************************************************
*    MAIN PROGRAM, TEST NODE     REVISED MAY 6, 1985
*    CALLS DMINIT, INPUT, EVENTS, OUTPUT, RESTOR
*****************************************************************
*INITIALIZE I/O UNITS
*INITIALIZE DYNAMIC MEMORY
 RESTART RUN
*INPUT SCENARIO AND TIME ZERO DATA
 INPUT RULES AT TIME ZERO
*SIMULATE EVENTS
*PRODUCE FINAL REPORTS AND SAVE STATUS
```

```
SUBROUTINE ALOCAT
*******************************************************
*  ALLOCATE OUTPUT TO LINKS USING CAPACITY AND DIRECT
*  COMMUNICATIONS ONLY
*******************************************************
*  CALLS - FIND, ERROUT, SNAP
*******************************************************
GET FIRST NODE
DO FOR ALL NODES
   DO FOR EACH MESSAGE ON FUTURE QUEUE
      CHECK TIME
         MOVE MESSAGE TO HOLD QUEUE
            ERROR IN ROUTING
         DESTINATION FOUND
         FIND LINK TYPE
            ERROR IN ROUTING
   GET NEXT MESSAGE
   DO FOR ALL DESTINATIONS
      DO FOR ALL LINKS
         FIND LAST SEND MESSAGE
         LAST MESSAGE FOUND
         MOVE HOLD QUEUE
         LOCATE LAST SEND MESSAGE IN CAPACITY
            LAST MESSAGE FOUND
            MOVE REMAINING MESSAGES TO HOLD QUEUE
         END OF SEND QUEUE CORRECTION
      GET NEXT LINK
   GET NEXT DESTINATION
   END OF SEND QUEUE JUSTIFICATION
GET NEXT NODE
LAST NODE COMPLETED
```

```
SUBROUTINE ALTOUT
*********************************************************
*  REVIEW MESSAGES IN HOLD QUEUES TO SEE IF THEY SHOULD
*  BE SENT VIA ALTERNATIVE DESTINATIONS
*********************************************************
*  CALLS MOVMSG, SNAP, FIND
*********************************************************
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      SET ALLOCATION FLAG OFF
      CHECK NODES ALTERNATES TO THIS DESTINATION
         DO FOR ALL LINKS TO THIS DESTINATION
            DO FOR ALL MESSAGES ON THIS HOLD QUEUE
               MOVE MESSAGES FROM PHOLD TO ALTERNATE1
               BY FIRST MESSAGE ALTERNATE
               RECHECK HOLD QUEUE
               CHECK IF ALTERNATE 2 EXISTS
               MOVE MESSAGES FROM PHOLD TO ALTERNATE 2
               BY FIRST MESSAGE ALTERNATE
               RECHECK HOLD QUEUE
               MOVE MESSAGES FROM PHOLD TO ALTERNATE 1
               BY SECOND MESSAGE ALTERNATE
               CHECK IF ALTERNATE 2 EXISTS
               RECHECK HOLD QUEUE
               MOVE MESSAGES FROM PHOLD TO ALTERNATE 2
               BY SECOND MESSAGE ALTERNATE
            GET NEXT LINK
      GET NEXT DESTINATION
   GET NEXT NODE
LAST NODE
```

```
SUBROUTINE CONTRL
**********************************************************
* READ IN MODE OF OPERATION AND PRINT CONTROL FLAGS
*    PRINT FLAG (PFLAG(I)) DEFINITIONS
*      1    ALL MESSAGES AT ALTERNATE DESTINATION
*      2    ALL MESSAGES ON INPUT QUEUES
*      3    ALL MESSAGES ON OUTPUT QUEUES
*      4    ALL MESSAGES ON FUTURE QUEUES
*      5    ALL MESSAGES BEING HELD
*      6    ALL MESSAGES DELETED
*      7    STATUS OF RULE STRUCTURE
*      8    CAS TAKE OFF SCHEDULED
*    9-10   NOT ASSIGNED
*     11    TRACKED MESSAGES AT ALTERNATE DESTINATIONS
*     12    TRACKED MESSAGES ON INPUT QUEUES
*     13    TRACKED MESSAGES ON OUTPUT QUEUES
*     14    TIME T OUTPUT ON FILE 14 REQUIRED
*     15    COMBAT LOSS VECTORS
*     16    FORCE RATIO CALCULATIONS
*     17    RULE STATUS AT FINAL TIME
*     18    NOT ASSIGNED
*     19    RANDOM PROCESSING REQUIRED
*     20    USED INTERNALLY FOR SUM OF FLAGS
********************************************************
*    PRINT MODIFIER (PMOD(I)) DEFINITIONS
*      1    OPTIONAL OUTPUT RESTRICTED TO THIS NODE
*      2    OPTIONAL OUTPUT STARTS AT THIS TIME
*      3    OPTIONAL OUTPUT STOPS AFTER THIS TIME
**********************************************************
```

```
 SUBROUTINE DMINIT(MEMORY,MPTR,IGBPTR,MAXDM)
*************************************************************
*   INITIALIZE DYNAMIC MEMORY
*************************************************************
*INITIALIZE MEMORY POINTER
*INITIAIZE GARBAGE POINTER
*CLEAR DYNAMIC MEMORY
```

```
SUBROUTINE ERROUT(MESAGE)
*************************************************************
*   PRINT OUT ERROR CONDITIONS AND STOPS EXECUTION
*************************************************************
```

```
SUBROUTINE EVENTS
*****************************************************
*   PROCESS SIMULATION EVENTS FROM CURRENT TIME TO END
*     OF RUN TIME
*   CALLS MSGIN, NODE, COMBAT, AIRBAS, TIMOUT
*****************************************************
INITIALIZE CMDRS STATUS BLOCK
START OF EVENTS IN A TIME INCREMENT
   PROCESS TIME T INPUT
   PROCESS EXTERNAL MESSAGES
   PROCESS NETWORK
   PROCESS AIRBASE
   PROCESS COMBAT
   CREATE OUTPUT
   PRODUCE TIME T OUTPUT IF REQUIRED
   TEST FOR LAST TIME INCREMENT
INCREMENT GAME TIME
END OF GAME TIME
```

```
SUBROUTINE EXTMSG
*********************************************************
*  GET MESSAGE FROM USER INPUT FOR CURRENT TIME
*  PUT MESSAGE ON ORIGINATORS FUTURE QUEUE
*  INPUT FOR MESSAGE
*     SEND TIME
*     MESSAGE TYPE
*     ORIGINATOR UNIT TYPE
*     DESTINATION UNIT ID
*     MESSAGE CREATION TIME
*     TRACKING FLAG
*     ATO FLAG
*     MESSAGE PRIORITY
*     TIME IN NETWORK
*  **   ATO DATA
*         COMBAT UNIT ID
*         EARLIEST TIME ON TARGET
*         LATEST TIME ON TARGET
*         AIRCRAFT TYPE
*         NUMBER OF AIRCRAFT
*         TYPE REQUEST, 0-PREPLAN, 1-IMMEDIATE
*********************************************************
 SKIP OVER HEADER DOCUMENTATION
GET FIRST MESSAGE
CHECK FOR CURRENT TIME
   PROCESS THIS MESSAGE
   CHECK FOR ADDITIONAL DATA
      GET ADDITIONAL DATA
      END ADDITIONAL DATA
   FIND NODE OF DESTINATION
   PUT ON INPUT QUEUE
   GET NEXT EXTERNAL MESSAGE
END PROCESSING THIS TIME FRAME
```

```
SUBROUTINE EXTSPT(PMSG,PDATA)
*****************************************************************
*  EXTSPT GETS THE DATA SECTIONS OF EXTERNAL MESSAGES 3136
*  AND PLACES THEM INTO THE DATA POINTER OF THE MESSAGE
*****************************************************************
GET FIRST INTEL REPORT
GET ADDITIONAL DATA ELEMENTS OF THE REPORT
```

```
SUBROUTINE FIND(PIN, N, ID, POUT)
*************************************************************
*  FIND A POINTER IN A QUEUE
*************************************************************
*  INPUT
*      PIN - POINTER TO TOP OF QUEUE TO BE SEARCHED
*      N   - OFFSET FROM PIN TO COMPARE
*      ID  - VALUE TO MATCH WITH N
*************************************************************
*  CREATES
*      POUT - POINTER TO DESIRED ELEMENT
*************************************************************
DO FOR ALL QUEUE ELEMENTS
   COMPARE VALUES
GET NEXT ELEMENT
END OF SEARCH
```

```
SUBROUTINE GIMME(NPTR, LEN, ISPACE)
*************************************************************
*   PROVIDE A BLOCK OF STORAGE FROM DYNAMIC MEMORY
*************************************************************
*   INPUT
*     LEN    - LENGTH OF BLOCK
*     ISPACE - ARRAY THAT CONTAINS DYNAMIC MEMORY
*************************************************************
*   OUTPUT
*     NPTR   - POINTER TO START OF ALLOCATED BLOCK
*************************************************************
*SEGMENT GET VIRTUAL SPACE
*IOUT IS OUTPUT DEVICE
*SEARCH GARBAGE LIST
*DO UNTIL LIST ENDS
    *IF (SIZE .EQ. LENGTH) THEN
      *SET PTR TO FIRST BLOCK
      *SNAP GARBAGE PTR
*ALLOCATE VIRGIN STORAGE
*UPDATE VIR SPACE PTR
*STORAGE OVRFLOW
*ZERO SPACE BLOCK
*END SEGMENT
```

```
SUBROUTINE HOLDQ1 (IPASS)
*************************************************************
*  MOVE PRIORITY MESSAGES FROM HOLD QUEUE TO ALTERNATE
*  COMMUNICATIONS LINK1 SEND QUEUE
************************************************************
*  CALLS - SNAP, FIND
************************************************************
*  INPUT
*     IPASS - FLAG FOR SELECTION OF DESTINATION
************************************************************
DO FOR ALL NODES
DO FOR ALL DESTINATIONS
SET ALLOCATION FLAG OFF
   DO FOR ALL LINKS
      DO FOR ALL MESSAGES ON HOLD QUEUE
         SET UP FOR MULTIPLE DESTINATIONS
            GET ALTERNATE DESTINATION
            CHECK IF ALTERNATE EXISTS
               FIND DESTINATION STRUCTURE
         GET FIRST ALT LINK OF MESSAGE
         CHECK FOR 0 TYPE
         GET ALTERNATE LINK
         COMPARE LINK TYPES
            LINK TYPES MATCH
            CHECK ALTERNATE LINK HOLD QUEUE PRIORITY
            CHECK SEND QUEUE
                  HOLD MESSAGE HAS GREATER PRIORITY
                  CHECK CAPACITY
                      PUT HOLD MESSAGE ON SEND QUEUE
                     SET FLAG = 1
               GET NEXT SEND MESSAGE
            END OF SEND QUEUE
            GET NEXT LINK
         GET NEXT MESSAGE ON HOLD QUEUE
         END OF HOLD QUEUE
      GET NEXT LINK
      END OF FIRST LINK LOOP

      MOVE EXCESS MESSAGES FROM SEND TO HOLD QUEUES
      DO FOR ALL LINKS
         DO UNTIL LINK CAPACTITY USED
            GET CORRECT CAPACITY
               GET NEXT MESSAGE
         MOVE EXCESS MESSAGES TO HOLD QUEUES
            CHECK FOR ALTERNATE STATUS
               PLACE IN THIS LINK HOLD QUEUE
               END THIS MOVE
            PLACE IN ANOTHER QUEUE
            GET NEXT MESSAGE
         END MESSAGE MOVES
      GET NEXT LINK
   END OF LINK QUEUE
GET NEXT DESTINATION
END OF DESTINATION QUEUE
```

GET NEXT NODE
LAST NODE COMPLETED

```
SUBROUTINE HOLDQ2 (IPASS)
***********************************************************
*  MOVE PRIORITY MESSAGES FROM HOLD QUEUE TO ALTERNATE
*  COMMUNICATIONS LINK2 SEND QUEUE
***********************************************************
*  CALLS - SNAP, FIND
***********************************************************
*  INPUT
*     IPASS - FLAG FOR ALTERNATE DESTINATIONS
***********************************************************
DO FOR ALL NODES
DO FOR ALL DESTINATIONS
SET ALLOCATION FLAG OFF
   DO FOR ALL LINKS
      DO FOR ALL MESSAGES ON HOLD QUEUE
         SET UP FOR MULTIPLE DESTINATIONS
            GET ALTERNATE DESTINATION
            CHECK IF ALTERNATE EXISTS
               FIND DESTINATION STRUCTURE
         GET SECOND ALT LINK OF MESSAGE
         CHECK FOR 0 TYPE
         GET ALTERNATE LINK
         COMPARE LINK TYPES
            LINK TYPES MATCH
            CHECK ALTERNATE LINK HOLD QUEUE PRIORITY
            LINK TYPES MATCH, CHECK SEND QUEUE
                  HOLD MESSAGE HAS GREATER PRIORITY
                  CHECK CAPACITY
                      PUT HOLD MESSAGE ON SEND QUEUE
                     SET FLAG = 2
               GET NEXT SEND MESSAGE
            END OF SEND QUEUE
            GET NEXT LINK
         GET NEXT MESSAGE ON HOLD QUEUE
         END OF HOLD QUEUE
      GET NEXT LINK
      END OF FIRST LINK LOOP

      MOVE EXCESS MESSAGES FROM SEND TO HOLD QUEUES
      DO FOR ALL LINKS
         DO UNTIL LINK CAPACITY USED
            GET CORRECT CAPACITY
               GET NEXT MESSAGE
         MOVE EXCESS MESSAGES TO HOLD QUEUES
            CHECK FOR ALTERNATE STATUS
               PLACE IN THIS LINK HOLD QUEUE
               END THIS MOVE
            PLACE IN ANOTHER QUEUE
            GET NEXT MESSAGE
         END MESSAGE MOVES
      GET NEXT LINK
   END OF LINK QUEUE
GET NEXT DESTINATION
END OF DESTINATION QUEUE
```

GET NEXT NODE
LAST NODE COMPLETED

```
SUBROUTINE INPUT
*********************************************************
*  WORKS FOR T = 1 ONLY
*********************************************************
*  CALLS FIND, GIMME
*********************************************************
CHECK FOR TIME T DATA
   PUT TIME T DATA ON UNIT 11
NODE IN
CHECK IF NODE STRUCTURE EXISTS
   CREATE NEW NODE STRUCTURE
   INITIALIZE NEW NODE STRUCTURE
   CREATE MESSAGE COUNT STRUCTURE
CREATE DESTINATION STRUCTURE
CREATE STATUS STRUCTURE FOR SUBORDINATES
   SET COMMANDER ID
END NODE IN
INPUT NODE MESSAGE PROCESSING LIMITS
   CHECK FOR TIME T DATA
   PUT TIME T DATA ON UNIT 12
INPUT COMMUNICATIONS LINK DATA
   CHECK FOR TIME T DATA
   PUT TIME T DATA ON UNIT 13
PROCESS FOR EACH END OF LINK
FIND ORIGIN
FIND DESTINATION
CREATE LINK STRUCTURE
PUT LINK ON END OF QUEUE
   END OF LINK QUEUE FOUND
INITIALIZE LINK DATA
CHECK FOR SECOND PASS
SWAP NODES AND REPEAT PROCESS
END OF LINK IN
INITIALIZE OUTPUT MESSAGE PROCESS
PROC IN
INPUT COMBAT DATA
INPUT AIR OPERATIONS DATA
INITIALIZE ALL ALTERNATE DESTINATION POINTERS, DESTINATION
  UNIT TYPES, AND POINTER TO COMMANDERS

DO FOR ALL NODES
   REPLACE COMMANDER UNIT NUMBER WITH POINTER
   DO FOR ALL DESTINATIONS
      FILL IN DESTINATION UNIT TYPE
      REPLACE DESTINATION ALTERNATES WITH POINTERS
      CHECK ALL DESTINATIONS FOR BOTH ALTERNATES
      SET ALTERNATE 1
      SET ALTERNATE 2
      GET NEXT DESTINATION
   GET NEXT NODE
LAST NODE CHANGED
PREPARE TIME T FILES FOR USE
```

```
SUBROUTINE MOVMSG(PMSG, PMSGO, PDEST, IUNIT, IALT)
*********************************************************
*  MESSAGES IN THE PMSG QUEUE ARE COMPARED TO DESTINATION
*  UNIT TYPE AND TO MESSAGES IN THE PDEST > PSEND QUEUE
*  FOR LINK TYPE, PRIORITY AND CAPACITY.  APPROPRIATE
*  MESSAGES ARE MOVED AND LOWER PRIORITY MESSAGES ON THE
*  SEND QUEUE WILL BE BUMPED.
*********************************************************
*  INPUT
*     PMSG -  POINTER TO SOURCE QUEUE
*     PMSGO -  LOCATION OF PMSG
*     PDEST -  POINTER TO A DESTINATION STRUCTURE
*     IUNIT -  DESTINATION UNIT
*     IALT  -  ALTERNATE DESTINATION FLAG
*           -  FIRST ALT = 1,  2ND ALT = 2
*********************************************************
SET ALTERNATE FLAG
DO FOR ALL MESSAGES
   CHECK MESSAGE FOR NO ALTERNATE
      CHECK FOR DESTINATION UNIT
         UNIT OK, TRY EACH LINK
            GET LINK
            LINK TYPE MATCHES
            PUT MESSAGE ON SEND QUEUE BY PRIORITY
                  CHECK CAPACITY
                     PUT MESSAGE ON SEND QUEUE
                     RESET TRANSMISSION FLAG
               GET NEXT MESSAGE ON SEND QUEUE
               LINKS DONT MATCH, GET NEXT LINK
         END LINK QUEUE
      UNIT TYPE WRONG
   LAST DESTINATION
GET NEXT MESSAGE ON MESSAGE QUEUE
LAST MESSAGE PROCESSED
```

```
SUBROUTINE MSGIN(PMSG, POMP, PNODE)
********************************************************
*  MSGIN LOGS AN INPUT MESSAGE INTO EACH OUTPUT TYPE BY
*  ORIGINATOR AND INPUT TYPE, SAVES MESSAGE DATA AND
*  DELETES MESSAGE SPACE
********************************************************
*  CALLS GIMME, RELEAS
********************************************************
*  INPUT
*     PMSG - POINTER TO INPUT MESSAGE
*     POMP - POINTER TO DESTINATION NODE
*            OUTPUT MESSAGE TYPE QUEUE
********************************************************
GET INPUT MESSAGE TYPE
GET INPUT MESSAGE ORIGINATOR AND TYPE
DO FOR ALL OUTPUT TYPES
   GET FIRST INPUT POINTER
   DO UNTIL INPUT TYPE FOUND
      TEST MESSAGE TYPE
         TEST ORIGINATOR TYPE UNIT
            LOOK FOR EXISTING UNIT LOG
            GET NEXT LOG
            ORIGINATOR NOT ON LIST, CREATE ENTRY
             PUT RULE NUMBER IN DATA STRUCTURE

            SET FLAG AND AGE
            CHECK FOR OTIME; IF OLD END PROCESS
            SET MESSAGE TRACK FLAG
   NEXT INPUT TYPE
NEXT OUTPUT TYPE
LAST POMP, SAVE MESSAGE DATA
CASE (MESSAGE TYPE)
   AIR OPERATIONS
      PUT ATO ON ORDER QUEUE
      PUT ATO ON REQUEST QUEUE
         SET RETURN NODE POINTER
    SPOT LOSS REPORT
END MSGIN
```

```
SUBROUTINE NODE
*************************************************************
*  PROCESS C3 EVENTS
*  CALLS - FIND, ERROUT, SNAP, MSGIN, MSGOUT, ALOCAT,
*     ALTOUT, LIMIT, SEND
*************************************************************
PROCESS INPUT MESSAGES
OPTIONAL PRINT
 IF(PFLAG(20).EQ.0) GO TO 8
   ALTERNATE DESTINATIONS
   INPUT QUEUES
   GET FIRST NODE
   DO FOR ALL NODES
      GET NODE IDENT.
      LIMIT NUMBER OF INPUT MESSAGES
      DO FOR ALL MESSAGES ON INPUT QUEUE
         IF MESSAGE IS ADDRESSED TO ANOTHER NODE
            REROUTE MESSAGE, PUT ON HOLD QUEUE
               FIND DESTINATION LINK
               ERROR IN ROUTING
            DESTINATION FOUND, SNAP IN
            RESET ALTERNATE COMMUNICATION FLAG
            FIND LINK TYPE
            CHECK LINK EXISTANCE
               TRY FIRST ALTERNATE LINK
                  LINK MATCHES, MODIFY MESSAGE
               TRY SECOND ALTERNATE LINK
                  LINK MATCHES, MODIFY MESSAGE
                DELETE MESSAGE
         USE DECISION RULES FOR MESSAGE PROCESSING
      GET NEXT MESSAGE
   LAST MESSAGE, GET NEXT NODE
   RETURN ANY HELD INPUT MESSAGES TO IMQUEUE
LAST NODE
PROCESS DECISION RULES
   GET FIRST NODE
   DO FOR ALL NODES
      DO FOR ALL OUTPUT MESSAGE TYPES
      GET FIRST TYPE OUTPUT
         TEST FOR PERIODIC PROCESS
         CHECK FOR PERIODIC TIME
            TEST FOR FIRST TIME FOR PROCESS
         TEST FOR RANDOM PROCESS
         DO FOR ALL INPUT MESSAGE TYPES
         GET FIRST TYPE INPUT
         INITIALIZE FLAG SUM
            DO FOR ALL INPUT MESSAGES
            INCREMENT AGE
            TEST FOR SINGLE USE FLAG
               TEST FOR USE OF MESSAGE
            TEST AGE GREATER THAN LIMIT
               SET FLAG TO OLD
            SUM INPUT FLAGS
            TEST FOR EACH MESSAGE TO CREATE A PROCESS
```

```
            GET NEXT INPUT MESSAGE
            GET NEXT INPUT TYPE
         INPUT TYPES COMPLETE, TEST FOR OUTPUT
            OUTPUT ACTION REQUIRED
            OUTPUT ACTION IS PERIODIC
         GET NEXT OUTPUT TYPE
      GET NEXT NODE
LAST NODE
LIMIT MESSAGES PROCESSED
ALLOCATE OUTPUT TO LINKS
ADD ALTERNATE ROUTINGS
ADJUST LINKS TO LIMIT
SECOND ALTERNATE ROUTINGS
ADJUST LINKS TO LIMIT
FIRST ALTERNATE DESTINATION
   LINK2
SECOND ALTERNATE DESTINATION
   LINK 2
THIRD ALTERNATE ROUTING
OPTIONAL PRINT
   OUTPUT QUEUES
   FUTURE QUEUES
   HOLD QUEUES
SEND MESSAGES
END OF NODE PROCESSING
*DO UNTIL LIST ENDS
```

```
SUBROUTINE OUTPUT
****************************************************************
*  PRODUCES A SUMMARY OF MESSAGE TRAFFIC AND CLOSE AIR SUPPORT
****************************************************************
IOUT IS OUTPUT DEVICE
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      DO FOR ALL LINKS
         DO FOR ALL MESSAGES ON HOLD
         GET NEXT MESSAGE
      GET NEXT LINK
   GET NEXT DESTINATION
GET NEXT NODE
LAST NODE
```

```
      SUBROUTINE PMSG1(PF)
      **********************************************************
      * PRINT OUT ALTERNATE DESTINATION, INPUT AND FUTURE
      *  MESSAGES UNDER PRINT FLAG CONTROL
      *  INPUT
      *     PF = 1 ALTERNATE DESTINATION
      *          2 INPUT QUEUES
      *          4 FUTURE QUEUES
      **********************************************************
     IOUT IS OUTPUT DEVICE
                 PRINT ALL NODES
                    TEST FOR ALTERNATE DESTINATION
                 GET NEXT NODE
     TEST FOR ALL INPUT MESSAGES
        TEST FOR FIRST PRINT TIME
           TEST FOR LAST PRINT TIME
              TEST FOR SPECIFIC NODE PRINT
                 PRINT ALL NODES
                    TEST FOR ALTERNATE DESTINATION
                 GET NEXT NODE
              PRINT SPECIFIC NODE
                 TEST FOR ALTERNATE DESTINATION
     TEST FOR TRACKED MESSAGES
        PRINT TRACKED MESSAGES ONLY
           TEST FOR ALTERNATE DESTINATION
           TEST FOR TRACKING FLAG
           GET NEXT NODE
```

```
   SUBROUTINE PMSG2(PF)
   ***********************************************************
   * PRINT OUT ALTERNATE OUTPUT AND HOLD
   *  MESSAGES UNDER PRINT FLAG CONTROL
   *  INPUT
   *     PF = 3 OUTPUT QUEUES
   *          5 HOLD QUEUES
   ***********************************************************
  IOUT IS OUTPUT DEVICE
  TEST FOR ALL INPUT MESSAGES
     TEST FOR FIRST PRINT TIME
        TEST FOR LAST PRINT TIME
           TEST FOR SPECIFIC NODE PRINT
              PRINT ALL NODES
                    GET NEXT LINK
                 GET NEXT DESTINATION
              GET NEXT NODE
           PRINT SPECIFIC NODE
                    GET NEXT LINK
                 GET NEXT DESTINATION
  TEST FOR TRACKED MESSAGES
     PRINT TRACKED MESSAGES ONLY
                    GET NEXT LINK
                 GET NEXT DESTINATION
        GET NEXT NODE
```

```
SUBROUTINE RELEAS (NPTR,LEN,ISPACE)
*****************************************************************
*   SEGMENT RELEASE PUTS STORAGE ON GARBAGE LIST
*****************************************************************
*IOUT IS OUTPUT DEVICE
*CHECK BAD PTR, LEN
**DO UNTIL NO GARBAGE EQUAL LENGTH
*END DO
*SNAP IN SPACE
*GARBAGE LENGTH NOT KNOWN
*PUT STORAGE ON GARBAGE LIST
*END SEGMENT
```

```
SUBROUTINE SEND
********************************************************
*  MOVES MESSAGES FROM ORIGINATOR SEND QUEUE TO
*  DESTINATION INPUT QUEUE, UPDATES LINK CAPACITIES AND
*  DELETES OUT OF TIME MESSAGES ON HOLD QUEUES
********************************************************
*  CALLS - FIND, RELEAS
********************************************************
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      GET DESTINATION NODE
      DO FOR ALL LINKS
         INCREMENT MESSAGE SENT COUNTER
         MOVE CONTENTS OF SEND QUEUE TO DESTINATION INPUT
         MERGE CONTENTS OF SEND AND INPUT QUEUES
         FIND LAST MESSAGE ON SEND QUEUE
         LAST MESSAGE FOUND
         GET NEXT LINK
         DO FOR ALL MESSAGES ON HOLD
            CHECK AGE OF MESSAGE
               DELETE THIS MESSAGE
                  DELETE DATA STRUCTURE
               DELETE MESSAGE STRUCTURE
         GET NEXT MESSAGE
      GET NEXT LINK
   GET NEXT DESTINATION
GET NEXT NODE
LAST NODE
```

```
 SUBROUTINE SNAP(PTR, NWORD, PIN,ISPACE)
*****************************************************************
*  SUBROUTINE SNAP PLACES A RECORD IN SORTED ORDER IN A QUEUE
*****************************************************************
*  INPUT
*    PTR    - POINTER TO ROOT OF QUEUE
*    NWORD  - OFFSET IN RECORD STRUCTURE FOR SORT VALUE
*    PIN    - POINTER TO RECORD TO BE INSERTED
*    ISPACE - ARRAY THAT CONTAINS DYNAMIC MEMORY
*****************************************************************

*EMPTY QUEUE - MAKE FIRST RECORD


*INSERT BEFORE RECORD

*INSERT BEFORE 1ST RECORD

*INSERT AFTER LAST RECORD
```

```
SUBROUTINE SAVE
**********************************************************
* DYNAMIC MEMORY AND COMMON VALUES ARE WRITTEN TO A FILE
* AS PHYSICAL STRUCTURES. THIS FILE MAY BE USED TO
* RESTART THE SIMULATION AT THE POINT WHERE IT LEFT OFF.
**********************************************************
SAVE COMMON VARIABLES
SAVE DYNAMIC MEMORY
```

```
 SUBROUTINE TIMOUT(HEAD)
 *************************************************************
 * PRINT OUT EACH MESSAGE THAT HAS ITS OUTPUT FLAG SET
 *************************************************************
IOUT IS OUTPUT DEVICE
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      DO FOR ALL LINKS
         DO FOR ALL MESSAGES ON HOLD
         GET NEXT MESSAGE
      REPEAT FOR SEND QUEUE
      GET NEXT LINK
   GET NEXT DESTINATION
GET NEXT NODE
LAST NODE
```

```
SUBROUTINE RESTOR
**********************************************************
* DYNAMIC MEMORY AND COMMON VALUES ARE READ FROM A FILE
* INTO MEMORY AS PHYSICAL STRUCTURES. THIS FILE IS
* CREATED BY SUBROUTINE STORE.
**********************************************************
RESTORE COMMON VARIABLES
RESTORE DYNAMIC MEMORY
```

```
SUBROUTINE RULEIN
*************************************************************
* CREATES OUTPUT MESSAGE PROCEDURES AND REQUIRED INPUT
* MESSAGE TYPES FOR EACH NODE. ALSO CREATES THE GENERIC
* OUTPUT MESSAGES FOR EACH RULE. GETS DATA FROM
* COMMON/RULE/ SET BY BLOCK DATA RULES.
*************************************************************
INCLUDE READ RULE DATA
DO FOR EACH PROCESS RULE IN BLOCK DATA
CREATE OUTPUT MESSAGE QUEUE
   CHECK EXISTING QUEUES FOR THIS RULE NUMBER
   NEW RULE
   RULE EXISTS, SET POINTER & SKIP CREATE QUEUE
   FIND NEXT MESSAGE FOR THIS RULE
      PUT MESSAGE ON QUEUE
      GET MESSAGE STRUCTURE
      GENERIC MESSAGE COMPLETE, GET NEXT MESSAGE
   OUTPUT MESSAGE QUEUE COMPLETE
   DO FOR EACH NODE OF THIS TYPE
         GET RULE STRUCTURE
         DO FOR EACH MESSAGE REQUIRED FOR THIS RULE
            FIND NEXT INPUT MESSAGE FOR THIS RULE
            CREATE INPUT MESSAGE STRUCTURE
            GET AN INPUT MESSAGE STRUCTURE
            INITIALIZE INPUT MESSAGE QUEUE
         END OF INPUT MESSAGE PROCESSING
      GET NEXT NODE
   END OF NODES
   GET NEXT RULE
END OF RULES, PRINT RULES OUT
```

```
SUBROUTINE RDRULE
************************************************************
* THIS ROUTINE READS THE THREE SETS OF DATA THAT MAKE UP
* THE COMMMAND POST RULES.
************************************************************
READ IN THE PREAMBLE DOCUMENTATION
READ THE RULE DATA SET
   READ RULES
READ THE INPUT MESSAGE REQUIREMENTS
   READ INPUT MESSAGES
READ THE OUTPUT MESSAGES
   READ OUTPUT MESSAGES
END OF RULE INPUT
 ************************************************************
 * PRINT OUT RULES
 ************************************************************
 IOUT IS OUTPUT DEVICE
 DO FOR ALL NODES
    DO FOR ALL INPUT MESSAGE TYPES
       DO FOR ALL INPUT MESSAGE LISTS
       NEXT INPUT MESSAGE TYPE
    PRINT OUT GENERIC MESSAGES
       GET NEXT MESSAGE
    GET NEXT POMP
 GET NEXT NODE
 LAST NODE
```

```
SUBROUTINE RULPRT
**************************************************************
* ECHOES OUT THE INPUT RULES
**************************************************************
DO FOR ALL LEVELS
   DO FOR ALL PROCESSES
         DO FOR ALL INPUT MESSAGES
```

```
SUBROUTINE LIMIT
*********************************************************
* LIMITS THE MESSAGE TRAFFIC ON EACH LINK TO THE TWO WAY MAXIMUM
* CAPACITY
*********************************************************
DO FOR ALL NODES
   DO FOR ALL DESTINATIONS
      TEST IF LIMITING HAS BEEN DONE FOR THIS DESTINATION
         SET LINK RECEIVER
         SET RECEIVER'S DESTINATION
      DO FOR ALL LINKS
         INITIALIZE COUNTERS
         FIND RECEIVERS LINK
         DO UNTIL CAPACITY USED
         CHECK PRIORITY OF BOTH MESSAGES
            TOP OF SEND LOOP
               GET CAPACITY FOR MESSAGE
               CHECK OTHER END OF LOOP
                  GET CAPACITY FOR MESSAGE
               BOTH QUEUES ENDED IN CAPACITY
            LINK CAPACITY EXCEDDED
            CORRECT LINK CAPACITY USED
            MARK THE END OF BOTH SEND QUEUES
            PROCESS BOTH QUEUES TO HOLD QUEUES
            MOVE EXCESS MESSAGES TO HOLD QUEUES
               PRIORITY CHECK
                  PRIORITIES SAME, CHECK CAPACITY
                     PUT MESSAGE BACK ON SEND QUEUE
                  END OF CAPACITY RECHECK
               END OF EQUAL PRIORITY CHECK
               PLACE IN APPROPRIATE QUEUE
               FIND DESTINATION
               FIND LINK
               RESET FLAG
               GET NEXT MESSAGE
            END MESSAGE MOVES
            INITIALIZE FOR SECOND PASS
      GET NEXT LINK
   GET NEXT DESTINATION
   FLAG BOTH DESTINATION QUEUES AS COMPLETED
GET NEXT NODE
LAST NODE
```

```
SUBROUTINE POUT (MEMORY, NUM, LENGTH)
****************************************************************
*   SUBROUTINE POUT PRODUCES A SNAP SHOT OF PART OF DYNAMIC
*   MEMORY
****************************************************************
```

```
      SUBROUTINE MAP
************************************************************************
*   SUBROUTINE MAP COMPUTES THE FORCE RATIOS BETWEEN RED AND BLUE FORCES
*     DESCRIPTION OF INPUTS FOR RED FORCES ARE
*        NTRJ=NUMBER OF TYPES OF WEAPONS-RED
*        NAMER(J)=NAMES ASSOCIATED WITH EACH OF THE NTRJ WEAPONS
*        NTCR(J)=NUMBER OF TYPE-J RED WEAPONS (TYPICAL CASE)
*        NR(J)=NUMBER OF TYPE-J RED WEAPONS (ACTUAL)
*        ER(J)=ENGAGEMENTS, PER TIME PERIOD, FOR EACH RED TYPE-J WEAPON
*        ALTCRB(J,I)=ALLOCATION (TYPICAL-CASE) RED AGAINST BLUE
*        PKRB(J,I)=PROBABILITY THAT RED J KILLS BLUE I GIVEN ENGAGEMENT
*     DESCRIPTION OF INPUTS FOR BLUE FORCES ARE
*        NTBI=NUMBER OF TYPES OF WEAPONS-BLUE
*        NAMEB(I)=NAMES ASSOCIATED WITH EACH OF THE NTBI WEAPONS
*        NTCB(I)=NUMBER OF TYPE-I BLUE WEAPONS (TYPICAL CASE)
*        NB(I)=NUMBER OF TYPE-I BLUE WEAPONS (ACTUAL)
*        EB(I)=ENGAGEMENTS, PER TIME PERIOD, FOR EACH BLUE TYPE-I WEAPON
*        ALTCBR(I,J)=ALLOCATION (TYPICAL-CASE) BLUE AGAINST RED
*        PKBR(I,J)=PROBABILITY THAT BLUE I KILLS RED J GIVEN ENGAGEMENT
*        THE ALGORITHM IS SET UP TO ACCEPT NO MORE THAT  11  DIFFERENT W
*        TYPES.
************************************************************************
*       ALLOCATION IS COMPUTED TWICE, RED AGAINST BLUE AND BLUE AGAINST
*     COMPUTE THE KILL RATE MATRICES (ACTUAL CASE)
*       SET ENGAGEMENT RATES
*     COMPUTE K (ACTUAL CASE)
*     OUTPUT THE INPUTS
```

```
SUBROUTINE ALLOC(ALBCX,NBCY,NY,NTX,NTY,AX)
************************************************************
*  THIS ROUTINE COMPUTES THE ACTUAL CASE ALLOCATION MATRICES
************************************************************
ALLOCATION MATRICES ARE COMPUTED EXCLUSIVE OF METHOD SUBROUTINES.
```

```
SUBROUTINE COMBAT
************************************************************
*  INTERFACE TO METHOD 4 CALCULATION OF APP AND THEN
*  CALCULATES THE COMBAT DRAW DOWN
************************************************************
DO FOR ALL NODES
      CREATE CAS REQUEST IF FORCE RATIO REQUIRES
         CHECK FOR ACTIVE ATO IN EXISTANCE
            REQUEST CAS
            TEST FOR RANDOM DELAY INDICATED
            PUT MESSAGE ON FUTURE QUEUE IN TIME SEQUENCE
```

```
      SUBROUTINE EIGENV(RB,BR,M,N,VR,VB,ALAM)
*********************************************************************
*THIS ROUTINE COMPUTES THE EIGEN VALUES AND LAMBDA FOR METH5 AND 4
*********************************************************************
*
*     MAXIMUM NUMBER OF ITERATIONS TO FIND AN EIGENVECTOR IS MNIE
*     SMALLEST DIFFERENCE IN EIGENVALUES IS EFCE.
*     SMALLEST ALLOWABLE DENOMINATOR (TO AVOID DIVISION BY ZERO) IS EPSL
*
*
*
*
```

```
      SUBROUTINE INPUTC
******************************************************
*   INPUT COMBAT VALUES
*     LCMBT - TYPE UNIT IN COMBAT
*     MTIME - TIME INCREMENT FOR COMBAT CALCULATION
*     FRBCAS - FORCE RATIO, RED/BLUE LIMIT FOR CAS
*     NODE - UNIT NUMBER
*     POSR,POSB - POSTURE OF UNIT,0 = RESERVE, 1 = OFFENSE,
*                 2 = DEFENSE, 3 = RETREAT
*     NR,NB - NUMBER OF COMBAT SYSTEMS
*     V  - INITIALIZED TO 1. FOR EIGEN VALUE BEST GUESS
******************************************************
*    READ IN THE INPUTS AND PRINT THEM OUT
      READ IN PREAMBLE DOCUMENTATION
      READ IN GENERIC RED UNIT DATA
      READ IN DATA VALUES
```

```
SUBROUTINE KILLS(EX,PKXY,ALLXY,J,I,EAKXY)
*****************************************************************
*  THIS ROUTINE COMPUTES THE KILL RATE MATRICES
*****************************************************************
THE RATE WEAPON X KILLS WEAPON Y IS A PRODUCT OF ENGAGEMENTS,ALLOC
AND PROBABILITY OF KILL FOR THOSE WEAPONS.
```

```
      SUBROUTINE METH4
*****************************************************************
*     SUBROUTINE METH4 CALCULATES THE FORCE RATIO
*****************************************************************
*
*
*     SET EACH ELEMENT OF THE BEST GUESS ARRAY TO 1
*
*
```

```
      SUBROUTINE MPROD(A,B,N,M,L,R)
***********************************************************************
*   THIS SUBROUTINE MULTIPLIES TWO MATRICES.
***********************************************************************
```

```
      SUBROUTINE PRNT(NTX,NTY,NTCX,NX,NAMEX,NAMEY,VALX,PKXY,EX,
*************************************************************************
* THIS ROUTINE OUTPUTS HEADINGS AND CONTROLS THE PRINTING OF THE
*     VARIOUS VARIABLES
*************************************************************************
*     OUTPUT NUMBER OF WEAPON TYPES
*     OUTPUT NUMBER OF EACH TYPE OF WEAPON
*     OUTPUT THE TYPICAL AND ACTUAL CASE NUMBERS
*     OUTPUT THE AVERAGE NUMBER OF ENGAGEMENTS
*     OUTPUT TYPICAL CASE ALLOCATION
*     OUTPUT ACTUAL CASE ALLOCATION
*     OUTPUT THE PROBABILITIES OF KILL
```

```
      SUBROUTINE PRNT2(NAMEX,PARAM1,PARAM2,NT,FMT1,FMT2,TWO)
C**********************************************************************
C*   THIS ROUTINE IS USED TO PRINT 1 DIMENSIONAL ARRAYS
C**********************************************************************
C     LUPLIM IS THE TOTAL NUMBER OF ROWS TO PRINT
C    TWO IS USED TO SIGNAL WHEN NOT TO PRINT A SECOND ARRAY
```

```
      SUBROUTINE REPORT
      ******************************************************************
      * THIS ROUTINE OUTPUTS THE INPUTS AND SELECTED COMPUTED VALUES IN
      * A FORMAT
      ******************************************************************
*
*     OUTPUT THE TITLE
*     OUTPUT THE INDEX WEAPON
*     OUTPUT THE BLUE VALUES
*     OUTPUT THE RED VALUES
*     OUTPUT K (BLUE AND RED)
```

```
      SUBROUTINE TABLES(SIDEO,SIDED,NAMEX,NAMEY,NTX,NTY,PARAM)
*********************************************************
*   THIS ROUTINE IS USED TO OUTPUT 2 DIMENSIONAL ARRAYS
*********************************************************
*     LUPLIM IS THE NUMBER OF COLUMNS TO PRINT
```

```
SUBROUTINE AIROPS
*****************************************************
* REPRESENTS THE WOC AND CRC ACTIONS
*****************************************************
***************************************************************
MAKE READY QUEUES CURRENT
DO FOR ALL CRC'S
   DO FOR ALL WOC'S
      REDUCE READY QUEUE
       GET NEXT A/C READY QUEUE
    *****************************************************
    SCHEDULE MISSION TAKE OFFS
    CHECK AIR SUPPORT REQUESTS
       CHECK FOR LAST USABLE TIME FOR ATO
          FIND A/C TYPE
          CHECK ENOUGH AIRCRAFT AVAILABLE
             CALCULATE NUMBER ASSIGNED
             PUT ATO ON CRC'S LIST
             MAKE MESSAGE FOR CORPS
             PUT MESSAGE ON FUTURE QUEUE
             WRITE ACTION ON OUTPUT
          AIR REQUEST WILL NOT BE FILLED
          SET NUMBER OF AIRCRAFT TO ZERO
          REMOVE FROM WOC'S ATO LIST
          SEND MESSAGE TO CORPS
       GET NEXT ATO
       ATO USED CASE
       ATO NOT USED CASE
    END OF THIS WOC SCHEDULE ACTION, PRINT STATUS
    GET NEXT WOC
 LAST WOC FOR THIS CRC COMPLETED SCHEDULE
 SET CAS FOR THIS TIME CYCLE
 DO FOR ALL NODES
    CHECK FOR COMBAT TYPE UNIT
       SET CAS SORTIES TO ZERO
    GET NEXT NODE
 CHECK FOR WOC SOURCE
 CHECK TIME ON TARGET
    SET CAS SUPPORT
    INCREMENT CAS SORTIE COUNT
    RESET UNIT AIR REQUEST TO NULL
    SCHEDULE SORTIE LANDING
    FIND WOC
    GET NEXT ATO
 LAST CAS SUPPORT PUT IN COMBAT, GET NEXT CRC
LAST CRC SCHEDULED
```

```
SUBROUTINE MAKMSG(MTYPE,ISEND,PNODE,IDEST,I1,I2,I3,ICAP,
*********************************************************
* MAKE ATO MESSAGES
* INPUT
*    MTYPE    MESSAGE NUMBER
*    ISEND    UNIT TYPE OF ORIGINATOR
*    PNODE    POINTER TO WOC'S NODE
*    IDEST    DESTINATION UNIT
*    I1-I3    COMMUNICATION LINK TYPES
*    ICAP     COMMUNICATION CAPACITY REQUIRED
*    PATO     POINTER TO AIR TASKING ORDER
*    IORIG    ORIGINATOR UNIT (98 IS INTERNAL)
* OUTPUT
*    PMSG     POINTER TO MESSAGE
* *******************************************************
* OTHER MESSAGE ELEMENTS SET
*    CREATE TIME TO NOW
*    MAXIMUM AGE TO 3
*    PRIORITY TO 1
*    OUTPUT FLAG IS ON
********************************************************
IF NO RETURN NODES, USE SUPPORT UNIT
GET ALTERNATES FOR MESSAGE
```

```
SUBROUTINE PRTATO(TITLE,PATO)
*******************************************************
* PRINTS AN AIR TASKING ORDER
*******************************************************
```

```
SUBROUTINE STATUS(TY1S, ICOUNT, PREADY)
*********************************************************
* PRINT WOC AIRCRAFT STATUS
*********************************************************
```

```
    SUBROUTINE INPUTA
    ***********************************************************
    *   INPUT AIR OPERATIONS VALUES
    *   ITYPE - TYPE OF INPUT LINE (CRC,'    ',END)
    *   NODEC - CRC UNIT NUMBER
    *   ATIME - ALERT RESPONSE TIME
    *   ETIME - TIME FROM TAKE OFF TO TARGET
    *   MINAC - MINIMUM NUMBER OF AIRCRAFT ON MISSION
    *   PS    - PROBABILITY OF SURVIVAL OF CAS AIRCRAFT
    *   NODEW - WOC UNIT NUMBER
    *   ACTYPE - AIRCRAFT TYPE
    *   ACTIME - TIME AIRCRAFT READY FOR TAKE OFF
    *   NUMAC - NUMBER AIRCRAFT READY FOR TAKE OFF
    *   COMMENT - DOCUMENTATION AND REVIEW
    **********************************************************
*  READ IN THE INPUTS AND PRINT THEM OUT
    READ IN THE PREAMBLE DOCUMENTATION
    READ IN HEADER DOCUMENTATION
       FIND NODE
       CREATE STRUCTURE FOR ALLOC
       STORE PARAMETERS IN ALLOC
    READ IN THE HEADER DOCUMENTATION
    GET NEXT DATA LINE
       CREATE CRC
       READY STRUCTURE
       GET NEXT AIRCRAFT ENTRY
       GET AIRCRAFT READY BLOCK
          MAKE NEW READY BLOCK
          CREATE RDYQ BLOCK
       END OF INPUT FOR AIR OPERATIONS
   DEBUG PRINT OF CRC AND WOC STRUCTURES
```

```
SUBROUTINE MINLIM(PNODE,MFLAG)
****************************************************************
*  SUBROUTINE MINLIM LIMITS THE NUMBER OF MESSAGES THAT MAY
*  ENTER A NODE AT ANY ONE TIME
****************************************************************
ZERO OUT MESSAGE IN COUNTERS
COUNT MESSAGS BY PRIORITY
   DON'T COUNT ALTERNATE MESSAGES
   DETERMINE CUT OFF PRIORITIES
      REMOVE MESSAGES FROM IMQ BY LOW PRIORITY
         DO NOT MOVE ALTERNATE MESSAGES
         DO NOT MOVE MESSAGE WITH ACCEPTABLE PRIORITY
            REMOVE THIS MESSAGE
            CHECK MESSAGE OUT OF AGE
            CHECK FOR MESSAGE PRIORITY BELOW DELETE LEVEL
               DELETE THIS MESSAGE
               CHECK FOR MEAASGE DATA
                  RETURN DATA SPACE
               RELEASE MESSAGE SPACE
            PUT MESSAGE ON HOLD QUEUE
            NEXT MESSAGE WHEN KEEPING MESSAGE ON IMQ
            GET NEXT MESSAGE
         END OF IMQ
RETURN MESSAGES ON HOLD TO IMQ
```

```
SUBROUTINE TINPUT
***********************************************************
* INPUT CHANGES TO CHARACTERISTICS DURING TIME T
* USES UNFORMATTED TEMPORARY FILES 11, 12, 13
***********************************************************
   GET FIRST MESSAGES
CHECK FOR CURRENT REINFORCEMENTS
CHECK FOR CURRENT NODE CHAGES
CHECK FOR CURRENT LIMIT CHANGES
CHECK FOR CURRENT LINK CHANGES
CHECK FOR CURRENT ALLOCATION PARAMETER CHANGES
   FIND NODE
   FIND ALLOCATION STRUCTURE FOR TYPE MSG
END TIME T INPUT FOR THIS TIME INCREMENT
```

```
SUBROUTINE CHFORC(NRC,NBC)
*******************************************************
*   INPUT COMBAT VALUES IN NRC AND NBC
*     1 - TIME FOR UNIT REINFORCEMENTS OR POSTURE
*         CHANGE
*     2 - UNIT NUMBER
*     3 - POSTURE OF UNIT,0 = RESERVE, 1 = OFFENSE,
*                           2 = DEFENSE, 3 = RETREAT
*     4-14 - NUMBER OF COMBAT SYSTEM REINFORCEMENTS
*******************************************************
```

```
SUBROUTINE CHLIM(LIMIT)
*****************************************************
* CHANGE INPUT OR OUTPUT MESSAGE LIMITS AT A NODE
*****************************************************
FIND NODE
```

```
SUBROUTINE CHNODE(NODET)
*****************************************************
* CHANGE COMMANDER OR SUBORDINATE NODE
*****************************************************
```

```
      SUBROUTINE CHLINK(LINKT)
******************************************************
* CHANGE LINK CAPACITIES DURING TIME T
******************************************************
```

```
 SUBROUTINE MOULIM
*************************************************************
* LIMITS GENERATION OF OUTPUT MESSAGES AT EACH NODE BASED ON
* THE NUMBER OF MESSAGES REQUIRED TO BE PROCESSED AT THIS NODE
* AND MESSAGE PRIORITY.
*      CALLED BY:  SUBROUTINE NODE AFTER PROCESSING DECISION
*                  RULES BUT BEFORE MESSAGE PATH ALLOCATION.
*************************************************************
 DO FOR ALL NODES
      DO FOR EACH MESSAGE ON FUTURE QUEUE
         CHECK TIME
            MOVE MESSAGE TO HOLD QUEUE
               ERROR IN ROUTING
            DESTINATION FOUND
            FIND LINK TYPE
               ERROR IN ROUTING
      GET NEXT MESSAGE
**************************************************
    CLEAR COUNTER ARRAY
    DO FOR ALL DESTINATIONS
       DO FOR ALL LINKS
          COUNT APPLICABLE MESSAGES ON HOLD QUEUE
             DON'T COUNT ALTERNATE MESSAGES
             DON'T COUNT MESSAGES TO BE SENT IN THE FUTURE
          COUNT APPLICABLE MESSAGES ON SEND QUEUE AND MOVE ALL
             MESSAGES FROM IT TO THE HOLD QUEUE
             DON'T COUNT ALTERNATE MESSAGES
              IF(MEMORY(PNODE+1).EQ.12)
             DON'T COUNT MESSAGES TO BE SENT IN THE FUTURE
    DETERMINE CUT OFF PRIORITIES
       DELAY OR DELETE MESSAGES
       COMMANDER AS DESTINATION FIRST
       NEXT PROCESS SUBORDINATE DESTINATIONS
       FINALLY PROCESS OTHER DESTINATIONS
    TEST FOR OUTPUT
    GET NEXT NODE
```

```
 SUBROUTINE MDELAY(PNODE,PFMQ,PDEST,JP1,JP2,JC1,JC2,
*****************************************************************************
* DETRMINES WHICH MESSAGE WILL BE SENT, HELD (SET SEND TIME TO
* NEXT TIME) OR DELETED. PROCESSES HOLD QUEUE ON EACH LINK FOR
* THE DESTINATION.
*    CALLED BY SUBROUTINE MOULIM
*    PARAMETERS
*      PDEST - POINTER TO DESTINATION STRUCTURE
*      JP1   - MAXIMUM PRIORITY MESSAGE PROCESSED
*      JC1   - NUMBER OF MESSAGES AT LEVEL JP1 PROCESSED
*      JP2   - MINIMUM PRIORITY MESSAGE DELETED
*      JC2   - NUMBER OF MESSAGES AT LEVEL JP2 NOT DELETED
*      NHOLD - NUMBER OF MESSAGES HELD FOR FUTURE PROCESSING
*      NDEL  - NUMBER OF MESSAGES DELETED
*      INDEX - =0, PROCESS FIRST DESTINATION FOR A NODE
*              =1, PROCESS ADDITIONAL DESTINATION
*****************************************************************************
 PROCESS FIRST DESTINATION FOR A NODE
 DO EACH LINK
       PRIORITY MESSAGE TO BE SENT
          IGNORE ALTERNATE MESSAGES
             IGNORE FUTURE MESSAGES
                TEST FOR EQUAL PRIORITY
                   TEST ENOUGH MESSAGES AT LOWER PRIORITY
                TEST FOR MESSAGE LESS THAN HIGHER PRIORITY
                      DELAY THIS MESSAGE
                      MOVE TO FMQ
                DELETE THIS MESSAGE
                TEST FOR ADDITIONAL DATA
          INCREMENT BACK POINTER
          GET NEXT MESSAGE
```

```
SUBROUTINE MSGOUT (PNODE, POMP,POUT,PDATA,PLENTH)
********************************************************
*  MSGOUT GENERATES AN OUTPUT MESSAGE TO ALL DESIGNATED
*  DESTINATIONS
********************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
********************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO OUTPUT MESSAGE TYPE
*     PDATA - POINTER TO MESSAGE DATA
*     PLENTH- LENGTH OF BLOCK FOR MESSAGE DATA
********************************************************
   CHECK FOR COMMANDER ONLY
      COMMANDER ONLY, SET DESTINATION
   DO FOR EACH DESTINATION = DESTINATION TYPE
      CHECK DESTINATION TYPE
      GET NEXT DESTINATION
      DESTINATION FOUND
   CREATE OUTPUT MESSAGE
   ENTER DATA
   TEST FOR RANDOM CHANGE OF MESSAGE LENGTH INDICATED
   SET ALTERNATE DESTINATIONS
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
         SET SECOND ALTERNATE TO NODES FIRST ALTERNATE
   TRY SECOND MESSAGE ALTERNATE
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET SECOND ALTERNATE TO SECOND NODE ALTERNATE
   END OF ALTERNATE LOGIC
   CHECK FOR MESSAGE TRACKING FLAG
      SET TRACKING FLAG
   CHECK FOR MESSAGE SEND TIME
      PUT MESSAGE ON FUTRUE QUEUE BY TIME
   PUT MESSAGE ON SEND QUEUE BY PRIORITY
      GET SEND QUEUE
      FIND LINK TYPE
   ATTACH MESSAGE BLOCK
   FINISHED IF COMMANDER ONLY
   GET NEXT DESTINATION
END OF OUTPUT MESSAGES
```

```
SUBROUTINE PROCES (PNODE,POMP)
***********************************************************
*  SUBROUTINE PROCESS PERFORMS RESPONSE ACTIONS BASED ON
*  CONDITIONS OF PROCESS RULES HAVING BEEN MET
***********************************************************
*  INPUT
*    PNODE  - POINTER TO NODE
*    POMP   - POINTER TO OUTPUT TYPE
***********************************************************
TEST FOR RANDOM MESSAGE PROCESSES
   CASE(OUTPUT MESSAGE TYPE)
*TYPES(2900,3000,3400,7000,9990,9993,3126,3130,3136,OTHER)
*CASE(REQUEST AIR SUPPORT)
   NON ALLOCATION PROCESS
   ALLOCATION PROCESS
*CASE(APPROVED AIR SUPPORT)
*CASE(DELETE ATO)
   ATO REQUEST FLAG
*CASE(REQUEST HELOCOPTER SUPPORT)
*CASE(APPROVED HELOCOPTER SUPPORT)
*CASE(DELETE HTO)
*CASE(ACCEPT STATUS REPORT FROM SUBORDINATE)
*CASE(RECEIVE SPOT INTEL REPORT ON RED FOE)
*CASE(CREATE MESSAGES FOR OUTPUT)
CASE(RANDOM OUTPUT MESSAGE PROCESS)
*GET NEXT OUTPUT MESSAGE
*DELETE ALL ATO'S USED BY THIS PROCESS
```

```
SUBROUTINE ATODEL (PNODE, POMP)
********************************************************
*  ATODEL DELETES ALL ATO REQUEST AND REPLY BLOCKS THAT
*  HAVE ITS PROCESS NUMBER
********************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
********************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
********************************************************
SKIP ALL DELETIONS FOR THE WOC
DELETE REQUESTS
   TEST FOR THIS PROCESS NUMBER
DELETE REPLYS
   TEST FOR THIS PROCESS NUMBER
DELETE SPOT REPORTS
   TEST FOR THIS PROCESS NUMBER
```

```
SUBROUTINE ATOALO (PNODE, POMP,POUT)
*****************************************************************
*  ATOALO ALLOCATES SYSTEMS TO SUBORDINATES TO
*  SUPPORT REQUESTS, MESSAGE TYPES 2900, 3000, 3400
*****************************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
*****************************************************************
GET FIRST ATO FOR THIS NODE
INITIALIZE POINTER FOR INTERNAL ATO QUEUE
DO FOR EACH ATO MESSAGE
   IF PROCESS MATCHES ATO, CREATE MESSAGE
      FIND EXISTING ENTRY IN INTERNAL ATO QUEUE
         SUPPORT NODE NOT ON QUEUE, CREATE ENTRY
      CONSOLIDATE SUPPORT DATA IN EXISTING ENTRY
   GET NEXT ATO
CHECK ATOQ
   CALCULATE PERCEIVED FORCE RATIOS
   SORT SUBORDINATE STATUS STRUCTURES FORCE RATIO QUEUE
      NODE DOES NOT ALLOCATE THIS TYPE, PROCESS ALL ATO'S
      TEST FOR CORRECT SET OF ALLOCATION PARAMETERS
         NOT CORRECT SET OF ALLOCATION PARAMETERS, GET NEXT SET
   ALLOCATION PARAMETERS FOUND
   TEST FOR CURRENT ALLOCATION TIME PERIOD
   CALCULATE NUMBER OF SORTIES TO BE ALLOCATED
   ALLOCATE SORTIES AVAILABLE
   LOG ACTUAL NUMBER OF SORTIES ALLOCATED
```

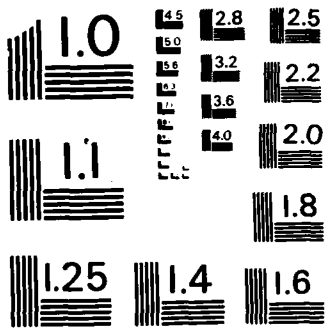
```
SUBROUTINE ALOCAS (PNODE,POMP,POUT,NALLOC,MINSOR,PATOQ)
*********************************************************
*  ALOCAS ALLOCATES AVAILABLE SORTIES TO REQUESTS FOR
*  CAS USING THE SUBORDINATES RED/BLUE FORCE RATIO QUEUE
*  TO SET PRIORITY
*********************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
*     NALLOC- NUMBER OF SORTIES TO ALLOCATE
*     MINSOR- MINIMUM OF SORTIES FOR A MISSION
*     PATOQ - POINTER TO TEMPORARY ATO ALLOCATION QUEUE
*********************************************************
DO WHILE SORTIES ARE AVAILABLE
   DO FOR EACH SUBORDINATE
         CHECK NUMBER OF SORTIES
         ADD SORTIES TO STATUS
      GET NEXT SUBORDINATE
   SEND REQUEST DENIED TO SUBORDINATES
         CREATE 7000 MESSAGE WITH ZERO SORTIES
      GET NEXT FRQ
   RESET NUMBER ALLOCATED TO ACTUAL NUMBER
   DELETE ALL ENTRIES IN TEMPORARY ATO QUEUE
```

```
SUBROUTINE PRATIO(PNODE)
*********************************************************
*   PRATIO CALCULATES SUBORDINATES RED/BLUE FORCE RATIO
*   AND ENTERS IT INTO THEIR STATUS STRUCTURE
*********************************************************
*   INPUT
*      PNODE - POINTER TO NODE
*********************************************************
GET STATUS QUEUE
DO FOR EACH SUBORDINATE STATUS STRUCTURE
   INITIALIZE RED FORCE ARRAY
   SUM PERCIEVED RED FORCES
   DO FOR EACH FOE
   TEST FOR FOE FORCES
      SET FORCE RATIO TO ZERO
   CALCULATE CURRENT FORCES
   CALCULATE FORCE RATIO
   SET FORCE RATIO CALCULATION TIME
   GET NEXT STATUS STRUCTURE
FINISHED FORCE RATIO CALCULATIONS
```

```
SUBROUTINE FQUEUE(PNODE)
***********************************************************
*  FQUEUE ORDERS THE SUBORDINATE'S RED/BLUE FORCE RATIO
*  QUEUE BY DESCENDING ORDER. THE QUEUE ROOT IS AT NODE+21
***********************************************************
*  INPUT
*     PNODE - POINTER TO NODE
***********************************************************
GET STATUS QUEUE
SET ALL FRQ POINTERS TO 5
FIND MAX FORCE RATIO
   GET NEXT PSTAT
CHECK IF MAX VALUE FOUND
   SET ENTRY AT END OF QUEUE
END OF ORDERING FORCE RATIO QUEUE
```

```
SUBROUTINE ATOOUT (PNODE, POMP,POUT)
*****************************************************************
*  ATOOUT CREATES OUTPUT MESSAGES THAT PASS ALONG AIR
*  SUPPORT REQUESTS, MESSAGE TYPES 2900, 3000, 3400
*****************************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
*****************************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
*****************************************************************
GET FIRST ATO FOR THIS NODE
   IF PROCESS MATCHES ATO, CREATE MESSAGE
         ADD SORTIES TO STATUS
```

AD-A168 973 C3 EVAL MODEL DEVELOPMENT AND TEST VOLUME 2 PROGRAMMERS 3/3
MANUAL..(U) INSTITUTE FOR DEFENSE ANALYSES ALEXANDRIA
VA R F ROBINSON ET AL. OCT 85 IDA-P-1882-VOL-2
UNCLASSIFIED IDA/HQ-85-30596 MDA903-84-C-0031 F/G 9/2 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
SUBROUTINE ATORTN (PNODE, POMP,POUT)
***********************************************************
*  ATORTN PASSES ALONG AIR SUPPORT APPROVALS TO THE
*  REQUESTOR USING MESSAGE TYPE 7000
***********************************************************
*  CALLS - GIMME, SNAP, FIND, ERROUT, RULES
***********************************************************
*  INPUT
*     PNODE - POINTER TO NODE
*     POMP  - POINTER TO OUTPUT TYPE
*     POUT  - POINTER TO MESSAGE TYPE
***********************************************************
   CHECK FOR ATO BLOCK
   CHECK ATO FOR THIS PROCESS
      GET AIR SUPPORT RETURN DESTINATION FROM ATO LIST
      FIND RETURN DESTINATION
   CREATE OUTPUT MESSAGE
   ENTER DATA
   SET ALTERNATE DESTINATIONS
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
         SET SECOND ALTERNATE TO NODES FIRST ALTERNATE
   TRY SECOND MESSAGE ALTERNATE
      SET FIRST ALTERNATE TO FIRST NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET FIRST ALTERNATE TO SECOND NODE ALTERNATE
   TRY SECOND NODE ALTERNATE
      SET SECOND ALTERNATE TO SECOND NODE ALTERNATE
   END OF ALTERNATE LOGIC
   CHECK FOR MESSAGE TRACKING FLAG
      SET TRACKING FLAG
   CHECK FOR MESSAGE SEND TIME
      PUT MESSAGE ON FUTRUE QUEUE BY TIME
   PUT MESSAGE ON SEND QUEUE BY PRIORITY
      GET SEND QUEUE
      FIND LINK TYPE
   GET NEXT ATO
RESET MESSAGE TRACKING FLAG OFF
```

B.3. Post-Processor

The post-processor consists of two programs. Both programs produce graphics from data files created by C^3EVAL. Program GraphSum uses files C^3SUM.DAT and LOSST.DAT for input to create summary graphs. Program GraphT uses files TIMET.DAT and LOSST.DAT for input to create time t graphs. File C^3SUM contains all summary data pertaining to the number of messages and sorties. File TIMET contains running totals for each time period for all data pertaining to the number of messages and sorties. File LOSST contains running totals for each time period for all data pertaining to the number of combat system losses.

B.3.a. Program GraphSum

GraphSum creates bar graphs using the summary data output by C^3EVAL. The actual graphing will be done by a call to VAXDECGRAPH. Any detailed information on DECGRAPH can be obtained from the appropriate VAXDECGRAPH manuals. There are 5 types of graphs: communications path limits, input message limits, output message limits, combat support and losses. The graphs represent the total messages/sorties/losses for a given time period for each uni' displayed. The first four graphs can display from 1 to 5 units at a time. The fifth graph can only display 1 unit at a time.

The first part of file C^3SUM is the preamble documentation. The first line of the preamble documentation is used as the subtitle for the graph. The last line of the preamble contains 'END' in card columns 1-4. Note that the preamble must contain at least 2 lines. After the 'END'card follows 3 header documentation lines. The rest of the file is data. There is a maximum of 19 units each represented by one data line of the form (1X,3A4,2I8,12I5). The values read are:

Unit identifier (consisting of 12 characters)
Unit number
Unit type
Communications limit in
Communications limit out
Communications limit held
Communications limit deleted
Input limit in
Input limit held
Input limit deleted
Output limit out
Output limit held
Output limit deleted

The last unit read in must be the unit number 1.

B.3.a(1). Subroutine GetSys

This subroutine is only called when the user requests to graph losses. The user is allowed to choose from 1 to 6 combat systems to graph from a list of 11 combat systems. The 11 combat systems are: apc, afv, tank, atank lt, atank, hv, mortar, artillery, helicopter, aaa, sam and cas. The user then chooses which unit to graph. The losses for the specified blue unit and the red units facing the blue unit are graphed. The units that the user can choose from are all units who have non-zero losses.

B.3.a(2). Subroutine GetVector

This subroutine determines which units will be graphed. There is a limit of 5 units per graph. The units to be graphed are determined at run time by the user. When the user selects the units, their locations within the data structure are stored in a unit vector for later recall. The units available for selection are determined by the input file.

B.3.a(3). Subroutine Graph1

Subroutine Graph1 creates the data file for a bar graph of the communications path limits. The data file consists of 2 sections: the instruction portion and the data portion. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each unit in the unit vector. Each line contains the unit name and the number of messages in, out, held and deleted by the communications links.

B.3.a(4). Subroutine Graph2

Subroutine Graph2 creates the data file for a bar graph of the input message limits. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each unit in the unit vector. Each line contains the unit name and number of messages in, held and deleted at the input side of the node.

B.3.a(5). Subroutine Graph3

Subroutine Graph3 creates the data file for a bar graph of the output message limits. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each unit in the unit vector. Each line contains the unit name and the number of messages out, held and deleted at the output side of the node.

B.3.a(6). Subroutine Graph4

Subroutine Graph4 creates the data file for a bar graph of the combat support. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label

label and vertical label for the graph. The data portion as a line for each unit in the unit vetor. Each line portion has a line for each unit in the unit vector. Each line contains the unit name and the number of close air support and helicopter sorties the unit received.

B.3.a(7). Subroutine Graph5

Subroutine Graph5 creates the data file for a bar graph of the losses. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The title contains the name of the unit being graphed. The data portion has a line for each combat system to be graphed. Each line contains the name of the combat system and the blue and red losses at the specified unit.

B.3.a(8). Subroutine Options

Subroutine Options allows the user to select at run time which type of graph is to be used. The six options available to the user are:

(1) Quit
(2) Communications Path Limits
(3) Input Limits
(4) Output Limits
(5) Combat Support
(6) Losses

B.3.a(9). Data Structures

NAME	TYPE	DESCRIPTION
COMSYS	A	1st dimension - multiple combat systems 2nd dimension 1-3 - name of the combat system 4 - combat system's number
LOSS	A	1st dimension - multiple time values 2nd dimension - multiple units 1-11 - blue combat systems losses 12-22 - red combat systems losses 23 - force ratio
NUMELEM	I	Number of units to graph
NUMSYS	I	Number of combat systems to graph
NUMUNIT	I	Number of units read in from data file
SCREEN	I	Value representing the action to take 1 - Quit 2 - Communications path limits 3 - Input limits 4 - Output limits 5 - Combat support 6 - Losses
SUBTI	A	40 character field for identifying the data set
SUMDATA	A	1st dimension - multiple time values 2nd dimension - multiple units 3rd dimension 1- 4 - Communication paths limits Number of messages in, out, held, and deleted 5- 7 - Input limits Number of messages in, held, and deleted 8-10 - Output limits Number of messages out, held, and deleted

NAME	TYPE	DESCRIPTION
		11-12 - Combat support Number of CAS and Helicopter sorties
UNITID	A	1st dimension - multiple units
		2nd dimension
		1- 3 - unit name
		4 - unit number
		5 - unit type
		6 - combat system losses flag
		0 --> unit suffered losses
		1 --> unit had no losses
VECTOR	A	The unit numbers of the units to graph

B.3.a(10). Program Notes

One possible enhancement would be to add the capability for the user to specify the range for the vertical axis. This way the graphs would have the same scales and could be placed side to side for comparison. The problem is that there seems to be no way to tell DecGraph to freeze the scale from the user's code. Another enhancement would be to specify colors for each part of the graph instead of leaving it to random selection.

B.3.a(11). Internal Code Documentation

PROGRAM GRAPHSUM

PURPOSE: CREATE BAR GRAPHS USING SUMMARY DATA OUTPUTTED BY C3EVAL. THERE ARE 5 TYPES OF GRAPHS: COMMUNICATIONS PATH LIMIT, INPUT LIMIT, OUTPUT LIMIT, COMBAT SUPPORT AND LOSSES.

LIMITATIONS:
MAXIMUM OF 19 UNITS BY DIMENSION.

EXTERNAL REFERENCES:
DECGRAPH

READ SUBTITLE

SKIP OVER THE PREAMBLE DOCUMENTATION

SKIP OVER THE HEADER DOCUMENTATION

READ IN THE SUMMARY DATA

READ LOSSES DATA FILE

INITIALIZE GRAPH DATA FILE

SELECT GRAPH

CHECK TO SEE IF USER IS READY TO QUIT

DETERMINE UNIT VECTOR AND EACH ELEMENTS LOCATION IN STORAGE

BRANCH TO OUTPUT APPROPRIATE GRAPH DATA

CREATE DATA FILE FOR GRAPH 1 - COMMUNICATIONS PATH LIMIT

CREATE DATA FILE FOR GRAPH 2 - INPUT LIMIT

CREATE DATA FILE FOR GRAPH 3 - OUTPUT LIMIT

CREATE DATA FILE FOR GRAPH 4 - COMBAT SUPPORT

CREATE DATA FILE FOR GRAPH 5 - LOSSES

CLOSE DATA FILE AND CREATE GRAPH

WAIT UNTIL USER READY TO CONTINUE

1 SUBROUTINE GETSYS(VECTOR,COMSYS,NUMSYS,UNITID,NUMUNIT)

PURPOSE:
USED WHEN GRAPHING LOSSES. DETERMINES WHICH COMBAT SYSTEM LOSSES TO GRAPH. EACH GRAPH IS LIMITED TO THE LOSSES AT A PARTICULAR NODE.

DETERMINE NUMBER OF COMBAT SYSTEMS TO GRAPH

DETERMINE WHICH COMBAT SYSTEMS TO GRAPH

DETERMINE WHICH UNIT TO GRAPH. ONLY UNITS WHO HAVE NON-ZERO LOSSES ARE SELECTABLE FOR GRAPHING.

STORE THE TITLFS OF THE COMBAT SYSTEMS THAT WERE CHOSEN TO BE GRAPHED.

1 SUBROUTINE GETVECTOR(VECTOR, NUMELEM, UNITID, NUMUNIT)

PURPOSE: DETERMINE WHICH UNITS WILL BE GRAPHED. FIND THE POSITION OF EACH UNIT WITHIN THE DATA STRUCTURE AND STORE THE LOCATIONS IN A VECTOR. SUBROUTINE GETVECTOR RETURNS THE LOCATION VECTOR AND THE NUMBER OF ELEMENTS IN THE VECTOR.

1 SUBROUTINE GRAPH1(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM)

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF COMMUNICATIONS PATH LIMITS. GRAPH HAS BARS FOR THE NUMBER OF MESSAGES IN, OUT, HELD, AND DELETED FOR EACH UNIT IN THE VECTOR.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE UNIT NAME IN QUOTES AND THE NUMBER OF MESSAGES IN, OUT, HELD AND DELETED.

1 SUBROUTINE GRAPH2(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM)

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF INPUT MESSAGE LIMITS. GRAPH HAS BARS FOR THE NUMBER OF MESSAGES IN, HELD, AND DELETED FOR EACH UNIT IN THE VECTOR.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS

OF THE UNIT NAME IN QUOTES AND THE NUMBER OF MESSAGES IN, HELD AND DELETED.

1 SUBROUTINE GRAPH3(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM)

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF OUTPUT MESSAGE LIMITS. GRAPH HAS BARS FOR THE NUMBER OF MESSAGES OUT, HELD, AND DELETED FOR EACH UNIT IN THE VECTOR.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE UNIT NAME IN QUOTES AND THE NUMBER OF MESSAGES OUT, HELD AND DELETED.

1 SUBROUTINE GRAPH4(SUBTI, UNITID, VECTOR, SUMDATA, NUMELEM)

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF COMBAT SUPPORT. GRAPH HAS BARS FOR THE NUMBER OF CLOSE AIR SUPPORT AND HELICOPTER SORTIES FOR EACH UNIT IN THE VECTOR.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE UNIT NAME IN QUOTES AND THE NUMBER OF CLOSE AIR SUPPORT AND HELICOPTER SORTIES.

1 SUBROUTINE GRAPH5(SUBTI,UNITID,VECTOR,LOSS,COMSYS,NUMSYS)

PURPOSE: CREATE DATA FILE FOR BAR GRAPH OF LOSSES. GRAPH HAS BARS FOR THE NUMBER OF BLUE AND RED LOSSES FOR EACH COMBAT SYSTEM FOR THE SPECIFIED UNIT.

CREATES INSTRUCTION PORTION OF DATA FILE

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE NAME OF THE COMBAT SYSTEM AND THE BLUE AND RED LOSSES AT THE SPECIFIED UNIT.

1 SUBROUTINE OPTIONS(SCREEN)

PURPOSE: ALLOWS THE USER TO SELECT WHICH TYPE OF GRAPH IS TO BE CREATED.

B.3.b. Program GraphT

GraphT creates line graphs using the summary data output by C^3EVAL. The actual graphing will be done by a call to VAXDECGRAPH. Any detailed information on DECGRAPH can be obtained from the appropriate VAXDECGRAPH manuals. There are 5 types of graphs: communications path limits, input message limits, output message limits, combat suport and losses. The graphs represent the number of messages/sorties/losses for a given unit for each time increment over a set time period.

The input file TIMET consists of a multiple number of data sets. Each data set represents the running totals for a given time period. Therefore, in order to obtain the number of messages/sorties for each time increment the previous total is subtracted from the current total. The first part of the file is the preamble documentation. The first line of the preamble documentation is used as the subtitle for the graph. The last line of the preamble contains 'END' in card columns 1 - 4. Note, The data sets follow the 'END' card.

Each data set starts with a line of the form (23X,I6) where the value is the time corresponding to the data. The second and third lines are header documentation lines. The rest of the data set is data. There is a maximum of 19 units each represented by one data line of the form (IX,3A4,2I8,12I5). The values read are:

Unit identifier (consisting of 12 characters)
Unit number
Unit type
Communications limit in
Communications limit out
Communications limit deleted
Input limit in
Input limit held

Input limit deleted
Output limit out
Output limit held
Output limit deleted

The last line read in for each data set must be the unit number 1.

B.3.b(1). Subroutine GetSys

This subroutine is only called when the user requests to graph losses. The user is allowed to choose from 1 to 6 combat systems to graph from a list of 11 combat systems. The 11 combat systems are: apc, afv, tank, atank, lt, atank, hv, mortar, artillery, helicopter, aaa, sam and cas. Then the user selects either the red or blue side to be graphed and which particular unit to graph. The units that the user can choose from are all units who have non-zero losses.

B.3.b(2). Subroutine GetUnit

This subroutine determines which unit will be graphed. The unit to be graphed is determined at run time by the user. When the user selects the unit its location within the data structure is stored for later recall. The units available for selection are determined by the input file.

B.3.b(3). Subroutine Graphl

Subroutine Graphl creates the data file for a line graph of the communications path limits at a specific node over a given time interval. The data file consists of 2 sections: the instruction portion and the data portion. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the time and the number

of messages in, out, held and deleted by the communications links for that one time increment.

B.3.b(4). Subroutine Graph2

Subroutine Graph2 creates the data file for a line graph of the input message limits at a specific node over a given time interval. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the tine and the number of messages in, held and deleted at the input side of the node for that one time increment.

B.3.b(5). Subroutine Graph3

Subroutine Graph3 creates the data file for a line graph of the output message limits at a specific node over a given time interval. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the time and the number of messages out, held and deleted at the output side of the node for that one time increment.

B.3.b(6). Subroutine Graph4

Subroutine Graph4 creates the data file for a line graph of the combat support at a specific node over a given time interval. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The data portion has a line for each time increment. Each line contains the tine and the number of close air support and helocopter sorties the node received for that one time increment.

B.3.b(7). Subroutine Graph5

Subroutine Graph5 creates the data file for a line graph of the losses at a specific node for either the red or blue side over a given time interval. The data file consists of 2 sections. The instruction portion contains the title, subtitle, horizontal label and vertical label for the graph. The title contains the name of the unit to be graphed and which side is to be graphed. The data portion has a line for each time increments. Each line contains the time and the number of losses for each combat system for the specified side at the specified node.

B.3.b(8) Subroutine Options

Subroutine Options allows the user to select at run time which type of graph is to be used. The six options available to the user are:

(1) Quit
(2) Communications Path Limits
(3) Input Limits
(4) Output Limits
(5) Combat Support
(6) Losses

B.3.b(9). Data Structures

NAME	TYPE	DESCRIPTION
COMSYS	A	1st dimension - multiple combat systems
		2nd dimension
		1-3 - name of the combat system
		4 - combat system's number
LOSS	A	1st dimension - multiple time values
		2nd dimension - multiple units
		3rd dimension
		1-11 - blue combat systems losses
		12-22 - red combat systems losses
		23 - force ratio
NUMSYS	I	Number of combat systems to graph
NUMUNIT	I	Number of units read in from data file
SCREEN	I	Value representing the action to take
		1 - Quit
		2 - Communications path limits
		3 - Input limits
		4 - Output limits
		5 - Combat support
		6 - Losses
SIDE	I	Value representing which side to graph
		1 - Blue
		2 - Red
SUBTI	A	40 character field for identifying the data set
SUMDATA	A	1st dimension - multiple time values
		2nd dimension - multiple units
		3rd dimension -
		1- 4 - Communications paths limits Number of messages in, out, held, and deleted
		5- 7 - Input limits Number of messages in, held, and deleted

NAME	TYPE	DESCRIPTION		
			8-10	- Output limits Number of messages out, held, and deleted
			11-12	- Combat support Number of CAS and Helicopter sorties
TIME	I	Number of time increments read in		
TIMET	A	The value of each time increment read in		
UNIT	I	The unit number of the unit to graph		
UNITID	A	1st dimension - multiple units		
		2nd dimension		
			1- 3	- unit name
			4	- unit number
			5	- unit type
			6	- combat system losses flag 0 --> unit suffered losses 1 --> unit had no losses

B.3.b(10). Program Notes

One possible enhancement would be to add the capability for the user to specify the range for the vertical axis. This way the graphs would have the same scales and could be placed side to side for comparison. The problem is that there seems to be no way to tell DecGraph to freeze the scale from the user's code. Another enhancement would be to specify colors for each part of the graph instead of leaving it to random selection.

B.3.b(11). Post Processor Internal Documentation

PROGRAM GRAPHT

PURPOSE: CREATE LINE GRAPHS USING TIME T DATA OUTPUTTED BY C3EVAL. THERE ARE 4 TYPES OF GRAPHS: COMMUNICATIONS PATH LIMIT, INPUT LIMIT, OUTPUT LIMIT, AND COMBAT SUPPORT.

LIMITATIONS:
MAXIMUM OF 19 UNITS BY DIMENSION.
THE LAST LINE OF EACH TIME T DATA SET MUST BE THE UNIT WHOSE IDENTIFIER IS 1.

EXTERNAL REFERENCES:
DECGRAPH

INITIALIZE WORKING AREA

READ TIME T DATA FILE
TIME T DATA FILE CONSISTS OF MULTIPLE TIME T DATA SETS

SKIP OVER THE PREAMBLE DOCUMENTATION

READ IN THE TIME CORRESPONDING TO THIS TIME T DATA SET

READ IN TIME T DATA SET

VALUES READ IN ARE RUNNING TOTALS AT TIME T. WE WANT TO HAVE THE VALUES FOR EACH INDIVIDUAL TIME T INCREMENT. THEREFORE, SUBTRACT THE PREVIOUS TIME T TOTAL.

THE LAST RECORD IN A TIME T DATA SET IS THE UNIT WHOSE IDENTIFIER IS EQUAL TO ONE.

READ LOSSES DATA FILE

FIND LOCATION OF UNIT NAME WITHIN ARRAY UNITID

INITIALIZE GRAPH DATA FILE

SELECT GRAPH

CHECK IF USER IS READY TO QUIT

DETERMINE UNIT TO BE GRAPHED AND ITS LOCATION IN THE DATA STRUCTURE

BRANCH TO OUTPUT APPROPRIATE GRAPH DATA

CREATE DATA FILE FOR GRAPH 1 - COMMUNICATIONS PATH LIMIT

CREATE DATA FILE FOR GRAPH 2 - INPUT LIMIT

CREATE DATA FILE FOR GRAPH 3 - OUTPUT LIMIT

CREATE DATA FILE FOR GRAPH 4 - COMBAT SUPPORT

CREATE DATA FILE FOR GRAPH 5 - LOSSES

CLOSE DATA FILE AND CREATE GRAPH

WAIT UNTIL USER READY TO CONTINUE

1 SUBROUTINE GETSYS(UNIT,COMSYS,NUMSYS,UNITID,NUMUNIT,SIDE)

PURPOSE:
USED WHEN GRAPHING LOSSES. DETERMINES WHICH COMBAT SYSTEM LOSSES TO GRAPH. EACH GRAPH IS LIMITED TO THE LOSSES AT A PARTICULAR NODE ON EITHER THE RED OR BLUE SIDE.

DETERMINE NUMBER OF COMBAT SYSTEMS TO GRAPH

DETERMINE WHICH COMBAT SYSTEMS TO GRAPH

DETERMINE WHETHER TO GRAPH BLUE OR RED SIDE

DETERMINE WHICH UNIT TO GRAPH. ONLY UNITS WHO HAVE NON-ZERO LOSSES ARE SELECTABLE FOR GRAPHING.

STORE THE TITLES OF THE COMBAT SYSTEMS THAT WERE CHOSEN TO BE GRAPHED.

1 SUBROUTINE GETUNIT(UNIT, UNITID, NUMUNIT)

PURPOSE: DETERMINE WHICH UNIT WILL BE GRAPHED. FIND THE POSITION OF THE UNIT WITHIN THE DATA STRUCTURE.

1 SUBROUTINE GRAPH1(SUBTI,UNITID,UNIT,SUMDATA,TIMET,TIME)

PURPOSE: CREATE DATA FILE FOR LINE GRAPH OF COMMUNICATIONS PATH LIMITS. GRAPH HAS LINES FOR NUMBER OF MESSAGES IN, OUT, H..LD AND DELETED WITH RESPECT TO TIME.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE TIME AND THE NUMBER OF MESSAGES IN, OUT, HELD AND DELETED.

1 SUBROUTINE GRAPH2(SUBTI,UNITID,UNIT,SUMDATA,TIMET,TIME)

PURPOSE: CREATE DATA FILE FOR LINE GRAPH OF INPUT MESSAGE LIMITS. GRAPH HAS LINES FOR NUMBER OF MESSAGES IN, HELD AND DELETED WITH RESPECT TO TIME.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE TIME AND THE NUMBER OF MESSAGES IN, HELD AND DELETED.

1 SUBROUTINE GRAPH3(SUBTI, UNITID, UNIT, SUMDATA, TIMET, TIME)

PURPOSE: CREATE DATA FILE FOR LINE GRAPH OF OUTPUT MESSAGE LIMITS. GRAPH HAS LINES FOR NUMBER OF MESSAGES OUT, HELD AND DELETED WITH RESPECT TO TIME.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE TIME AND THE NUMBER OF MESSAGES OUT, HELD AND DELETED.

1 SUBROUTINE GRAPH4(SUBTI, UNITID, UNIT, SUMDATA, TIMET, TIME)

PURPOSE: CREATE DATA FILE FOR LINE GRAPH OF COMBAT SORTIES. GRAPH HAS LINES FOR NUMBER OF CLOSE AIR SUPPORT AND HELICOPTER SORTIES WITH RESPECT TO TIME.

CREATES INSTRUCTION PORTION OF DATA FILE.

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE TIME AND THE NUMBER OF CLOSE AIR SUPPORT AND HELICOPTER SORTIES.

1 SUBROUTINE GRAPH5(SUBTI,UNITID,UNIT,LOSS,COMSYS,NUMSYS,SIDE,

PURPOSE:
CREATE DATA FILE FOR LINE GRAPH OF LOSSES. GRAPH HAS LINES FOR THE NUMBER OF RED OR BLUE LOSSES FOR THE SPECIFIED COMBAT SYSTEMS AT THE SPECIFIED NODE.

CREATES INSTRUCTION PORTION OF DATA FILE

CREATES DATA PORTION OF DATA FILE. EACH DATA LINE CONSISTS OF THE TIME AND THE NUMBER OF LOSSES FOR EACH COMBAT SYSTEM FOR THE SPECIFIED SIDE AT THE SPECIFIED NODE.

SUBROUTINE OPTIONS(SCREEN)

PURPOSE: ALLOWS THE USER TO SELECT WHICH TYPE OF GRAPH IS TO BE CREATED.

```
SUBROUTINE STATIN
**********************************************************
*   STATIN PERFORMS INITIATION ACTIONS FOR THE COMMANDERS
*   PERCEPTION OF SUBORDINATE STRENGTHS AND COMBAT STATUS
**********************************************************
DO FOR ALL NODES
   DO FOR ALL STAT (SUBORDINATES) BLOCKS
      SET BLUE STRENGTHS
```

```
SUBROUTINE STATOU
**********************************************************
*  STATOU PRINTS OUT THE COMMANDERS
*  PERCEPTION OF SUBORDINATE STRENGTHS AND COMBAT STATUS
**********************************************************
DO FOR ALL NODES
   DO FOR ALL STAT (SUBORDINATES) BLOCKS
      GET BLUE STRENGTHS
      GET POSTURES
```

```
SUBROUTINE STATUP (PNODE,POMP)
***********************************************************
*  SUBROUTINE STATUP UPDATES COMMANDERS PERSEPTIONS OF
*  SUBORDINATES COMBAT STATUS VIA MESSAGES 3126 AND 3130
***********************************************************
*  INPUT
*    PNODE  - POINTER TO NODE
*    POMP   - POINTER TO OUTPUT TYPE
***********************************************************
TEST FOR SUBORDINATE STATUS STRUCTURE
GET FIRST SPOT REPORT
    IF REPORT MATCHES PROCESS, LOG DATA
       FIND SUBORDINATE'S STATUS STRUCTURE
          ERROR, NO SUBORDINATE STATUS STRUCTURE

       DETERMINE SIDE
          UPDATE BLUE LOSSES AND STRENGTHS
          UPDATE BLUE POSTURE
          UPDATA BLUE DATA TIME TO LATEST TIME

          UPDATE RED LOSSES
          UPDATE RED POSTURE
          UPDATA RED DATA TIME TO LATEST TIME

       GET NEXT SPOT REPORT
 ALL SPOT REPORTS PROCESSED FOR THIS RULE
```

```
SUBROUTINE INTLUP (PNODE,POMP,POUT)
************************************************************
*  SUBROUTINE INTLUP UPDATES COMMANDERS PERSEPTIONS OF
*  SUBORDINATES COMBAT FOES VIA MESSAGE 3136
************************************************************
*  INPUT
*     PNODE  - POINTER TO NODE
*     POMP   - POINTER TO OUTPUT TYPE
*     POUT   - POINTER TO OUTPUT MESSAGE TYPE
************************************************************
GET FIRST SPOT REPORT
    IF REPORT MATCHES PROCESS, LOG DATA
      TEST FOR SUBORDINATE STATUS STRUCTURE
       FIND SUBORDINATE'S STATUS STRUCTURE
          ERROR, NO SUBORDINATE STATUS STRUCTURE

      UPDATE EXISTING FOE ESTIMATE IF NEW DATA IS LATEST
      CHECK DATA CURRENCY
         DELETE CURRENT ESTIMATES
      ENTER NEW ESTIMATES
      DELETE THESE DATA STRUCTURES
       GET NEXT SPOT REPORT
 ALL SPOT REPORTS PROCESSED FOR THIS RULE
```

```
SUBROUTINE CASLOS(SHOTB)
*****************************************************************
*  CASLOS SUBTRACTS COMBAT LOSSES DUE TO CAS FROM
*  RETURNING AIRCRAFT
*****************************************************************
   DO FOR ALL WOC'S
      SET TIME
      CHECK AIRCRAFT TYPE
            FIND CORRECT TIME
                  GET NEXT READY QUEUE
               REDUCE NUMBER OF AIRCRAFT RETURNING
            ERROR (NO SORTIES RETURNING)
         INCREMENT PREADY
      INCREMENT PWOC
   INCREMENT PCRC
ERROR (NO TYPE 1 AIRCRAFT RETURNING)
```

```
SUBROUTINE RPTLOS(PNODE,BL,RL,POSB,POSR)
*************************************************************
*  RPTLOS CREATES SPOT LOSS REPORTS
*************************************************************
*  INPUT
*     PNODE   - POINTER TO NODE STRUCTURE
*     BL      - ARRAY OF BLUE WEAPON SYSTEM LOSSES
*     RL      - ARRAY OF RED  WEAPON SYSTEM LOSSES
*************************************************************
 INITIALIZE SPOT REPORT
INITIALIZE MULTIPLIER INDICES
TEST FOR RANDOM EFFECTS REQUIRED
   CREATE BLUE AND RED RANDOM NUMBERS
COPY LOSSES
COPY POSTURES
TEST FOR RANDOM MESSAGE DELAY REQUIRED
PUT MESSAGES ON THE FUTURE QUEUE IN TIME ORDER
```

```
SUBROUTINE RANMSG(PNODE,POMP)
GET COMMONS
**********************************************************************
*   RANMSG CREATES ALL THE MESSAGES FOR A NODE THAT ARE CLASSIFIED
*   AS RANDOM. (IE PROCESS NUMBERS 3800, 4800, 5800, 6800, 7800,
*   5900, AND 7900.
**********************************************************************
TEST FOR FIRST TIME TO INITIALIZE NODE'S RANDOM QUEUE
      GET NEXT POUT
TEST FOR TIME TO SEND RANDOM MESSAGES
      SET UP GENERIC MESSAGE FOR NEXT RANDOM TIME
GET NEXT RANDOM MESSAGE
```

```
BLOCK DATA FRATIO
*************************************************************
* DATA FOR THE GENERIC TYPE UNIT COMBAT DATA
* FOR BOTH RED AND BLUE FORCES
*************************************************************
ENGAGEMENT RATES
TYPICAL NUMBER OF TYPES IN A UNIT
CONTROL SIZES
ALLOCATION OF RED AGAINST BLUE
ALLOCTATION OF BLUE AGAINST RED
PK RED AGAINST BLUE
PK BLUE AGAINST RED
```

PROPOSED
DISTRIBUTION
IDA PAPER P-1882

C^3 EVAL MODEL DEVELOPMENT AND TEST
Volume II
Programmers Manual

75 Copies

	Copies
DEPARTMENT OF DEFENSE	
Office Joint Chiefs of Staff Washington, DC 20301-5000	
ATTN: Dr. Robert Fallon, Office of Director, Command Control and Communications Systems	12
Dr. William G. Lese, JAD	1
LTC Joseph M. Cummings, USA, JAD	2
CPT W. L. Butler, USN, J-3	1
LTC J. P. Morrison, USAF,J-4	1
CDR T. R. Sheffield, USN, J-5	1
Directorate of Information and Resource Management	20
Office of the Under Secretary of Defense for Research and Engineering Room 3D139, The Pentagon Washington, DC 20301	
ATTN: Mr. James Bain, C^3I	1
Director Defense Intelligence Agency Washington, DC 20301-6111	
ATTN: Mr. A. J. Straub, (Pompino Plaza 1023)	1
Office of the Secretary of Defense OUSDRE (DoD-IDA Managment Office) 1801 N. Beauregard Street Alexandria, VA 22311	
ATTN: Col T. L. Ricketts	1
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2

DEPARTMENT OF THE ARMY

Deputy Chief of Staff for Operations and Plans
Room 3C542, The Pentagon
Washington, DC 20310-0403

ATTN:	Maj W. E. Ward, USA	1
	Mr. Hunter Woodall, DCS/RDA	1

DEPARTMENT OF THE NAVY

Chief of Naval Operations
Department of the Navy
Room 4E549, The Pentagon
Washington, DC 20350

ATTN:	CDR D. L. McKinney, USN NOP	1

Commander
Naval Postgraduate School
Monterey, CA 93940

ATTN:	Prof Michael G. Sovereign, Chairman Command, Control and Communications	2

HQ, U. S. Marine Corps
Columbia Pike and Arlington Ridge Road
Arlington, VA 22204

ATTN:	LTC T. L. Wilkerson, Office of the Deputy Chief of Staff Plans, Policy and Operations (MD-P)	1

Commander
Naval Postgraduage School
Monterey, CA 93940

ATTN:	Prof Michael G. Sovereign Chairman, Command, Control and Communications	1

DEPARTMENT OF THE AIR FORCE

Deputy Chief of Staff, Operations, Plans & Readiness
Department of the Air Force
Washington, DC 20330

ATTN:	LTC M. H. Long, AFXOX	2

CIVILIAN CONTRACTORS

Applications Research Corporation
330 S. Ludlow Street
Dayton, OH 45402

ATTN:	Mr. Rodney B. Beach	1

Institute for Defense Analyses
1801 N. Beauregard Street
Alexandria, VA 22311

22

ATTN:	Gen W. Y. Smith	1
	Mr. S. J. Deitchman	1
	Mr. R. Pirie	1
	Mr. A. R. Barbeau	1
	Dr. William J. Schultis	1
	Dr. Harry Williams	1
	Mr. Robert F. Robinson	1
	Mr. Joseph W. Stahl	1
	Dr. J. Bracken	1
	Mr. J. L. Freeh	1
	Mr. E. Kerlin	1
	Mr. A. O. Kresse	1
	Dr. D. L. Ockerman	1
	Dr. E. S. Parkin	1
	Dr. F. R. Riddell	1
	Dr. V. A. Utgoff	1
	Miss E. Doherty	1
	Control and Distribution	5

END

DTIC

8-86